



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Brew Labs

05 October 2023



paladinsec.co



info@paladinsec.co

Table of Contents

| | |
|----------------------------------|----|
| Table of Contents | 2 |
| Disclaimer | 4 |
| 1 Overview | 5 |
| 1.1 Summary | 5 |
| 1.2 Contracts Assessed | 6 |
| 1.3 Findings Summary | 7 |
| 1.3.1 Global Issues | 8 |
| 1.3.2 BrewlabsERC20 | 8 |
| 1.3.3 BrewlabsFactory | 8 |
| 1.3.4 BrewlabsNFTDiscountManager | 9 |
| 1.3.5 BrewlabsPair | 9 |
| 1.3.6 BrewlabsRouter | 9 |
| 1.3.7 BrewlabsSwapFeeManager | 10 |
| 1.3.8 BrewlabsLibrary | 10 |
| 2 Findings | 11 |
| 2.1 Global Issues | 11 |
| 2.1.1 Issues & Recommendations | 11 |
| 2.2 BrewlabsERC20 | 12 |
| 2.2.1 Issues & Recommendations | 13 |
| 2.3 BrewlabsFactory | 14 |
| 2.3.1 Privileged Functions | 14 |
| 2.3.2 Issues & Recommendations | 15 |
| 2.4 BrewlabsNFTDiscountManager | 18 |
| 2.4.1 Privileged Functions | 18 |
| 2.4.2 Issues & Recommendations | 19 |
| 2.5 BrewlabsPair | 25 |
| 2.5.1 Privileged Functions | 25 |

| | |
|--------------------------------|----|
| 2.5.2 Issues & Recommendations | 26 |
| 2.6 BrewlabsRouter | 31 |
| 2.6.1 Privileged Functions | 31 |
| 2.6.2 Issues & Recommendations | 31 |
| 2.7 BrewlabsSwapFeeManager | 32 |
| 2.7.1 Privileged Functions | 33 |
| 2.7.2 Issues & Recommendations | 34 |
| 2.8 BrewlabsLibrary | 45 |
| 2.8.1 Issues & Recommendations | 45 |



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 advocacy 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.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Brew Labs on the BNB Smart Chain. 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 Summary

| | |
|------------------------------|---|
| Project Name | Brew Labs |
| URL | https://brewlabs.info/ |
| Platform | BNB Smart Chain |
| Language | Solidity |
| Preliminary Contracts | https://github.com/brewlabs-code/brewlabs-swap/commit/f5302b4dfedaa02f2e26a4089d7b9c08a90c0337 |
| Resolution 1 | https://github.com/brewlabs-code/brewlabs-swap/tree/7082264def34f601b3d86bcb47176c9be8305b86 |
| Resolution 2 | https://github.com/brewlabs-code/brewlabs-swap/tree/df57dd1422b3fd72c645a0a01247cf8c66420010 |

1.2 Contracts Assessed

| Name | Contract | Live Code Match |
|----------------------------|---|-----------------|
| BrewlabsERC20 | Dependency in BrewlabsFactory | ✓ MATCH |
| BrewlabsFactory | 0xFE2bF5fc2D131dB07C5Ef7076856FD7f342738fF | ✓ MATCH |
| BrewlabsNFTDiscountManager | 0xA681C75FF9976F8559e2Ca14e7b42BD9746A53d9 | ✓ MATCH |
| BrewlabsPair | 0x13bBDBB16DE07F4dc10c773061da92F4002Fde1E Also dependency in BrewlabsFactory | ✓ MATCH |
| BrewlabsRouter | 0x5c71e01556b01B6346a4435B145a15aA3A957D23 | ✓ MATCH |
| BrewlabsSwapFeeManager | Proxy 0x9dF9d5A7597cd4BF781d4FA9b98077376F6643AD Implementation 0x9dF9d5A7597cd4BF781d4FA9b98077376F6643AD | UNMATCHED |
| BrewlabsLibrary | Dependency in BrewlabsRouter | UNMATCHED |



1.3 Findings Summary

| Severity | Found | Resolved | Partially Resolved | Acknowledged | Failed Resolution |
|-----------------|-------|----------|--------------------|--------------|-------------------|
| ● Governance | 2 | 1 | - | 1 | - |
| ● High | 5 | 2 | 2 | - | 1 |
| ● Medium | 3 | 1 | 1 | 1 | - |
| ● Low | 9 | 8 | - | 1 | - |
| ● Informational | 9 | 5 | 3 | 1 | - |
| Total | 28 | 17 | 6 | 4 | 1 |

Classification of Issues

| Severity | Description |
|-----------------|--|
| ● Governance | Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example. |
| ● 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. |

1.3.1 Global Issues

| ID | Severity | Summary | Status |
|----|----------|--|------------|
| 01 | LOW | The fee rate denominator is inconsistent | ✓ RESOLVED |

1.3.2 BrewlabsERC20

| ID | Severity | Summary | Status |
|----|----------|--|------------|
| 02 | INFO | Typographical issues | ✓ RESOLVED |
| 03 | INFO | Lack of events for approval update within transferFrom | ✓ RESOLVED |

1.3.3 BrewlabsFactory

| ID | Severity | Summary | Status |
|----|----------|---|--------------|
| 04 | LOW | Blacklist mechanism can be circumvented | ACKNOWLEDGED |
| 05 | LOW | feeTo can not be set to address(0) | ✓ RESOLVED |
| 06 | INFO | Typographical issues | ✓ RESOLVED |

1.3.4 BrewlabsNFTDiscountManager

| ID | Severity | Summary | Status |
|----|----------|---|--------------|
| 07 | HIGH | Manipulation of discounts by external parties | PARTIAL |
| 08 | MEDIUM | discountOf can return an incorrect discount | ACKNOWLEDGED |
| 09 | LOW | The order of the discounts is not enforced | ✓ RESOLVED |
| 10 | INFO | Gas optimizations | PARTIAL |
| 11 | INFO | Typographical issues | ✓ RESOLVED |

1.3.5 BrewlabsPair

| ID | Severity | Summary | Status |
|----|----------|---|--------------|
| 12 | GOV | Owner fee can be set to 100% | ✓ RESOLVED |
| 13 | LOW | The remaining fee calculation rounds down unnecessarily | ✓ RESOLVED |
| 14 | LOW | Read-only reentrancy via getReserves on the FeeManager | ✓ RESOLVED |
| 15 | INFO | Typographical issues | ✓ RESOLVED |
| 16 | INFO | Gas optimization | ACKNOWLEDGED |

1.3.6 BrewlabsRouter

No issues found.

1.3.7 BrewlabsSwapFeeManager

| ID | Severity | Summary | Status |
|----|----------|---|--------------|
| 17 | GOV | Governance can steal unclaimed rewards and DOS claimings | ACKNOWLEDGED |
| 18 | HIGH | A malicious user can steal rewards | ✓ RESOLVED |
| 19 | HIGH | The notifyRewardAmount function is prone to reentrancy exploit | ✓ RESOLVED |
| 20 | HIGH | If a FeeManager is set, unset and set again, rewards may be stolen | PARTIAL |
| 21 | HIGH | Checks-effects-interactions pattern is not adhered to | FAILED |
| 22 | MEDIUM | The rewardDebt calculation rounds down allowing users to steal some rewards | PARTIAL |
| 23 | MEDIUM | The tokenOwner logic can become malicious in specific scenarios | ✓ RESOLVED |
| 24 | LOW | The storage gap is incorrectly calculated | ✓ RESOLVED |
| 25 | LOW | Use call instead of transfer within rescueWrongToken | ✓ RESOLVED |
| 26 | LOW | Lack of safeTransfer usage within notifyRewardAmount | ✓ RESOLVED |
| 27 | INFO | Typographical issues | PARTIAL |
| 28 | INFO | Gas optimizations | PARTIAL |

1.3.8 BrewlabsLibrary



No issues found.

2 Findings

2.1 Global Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

2.1.1 Issues & Recommendations



| | |
|-----------------------|---|
| Issue #01 | The fee rate denominator is inconsistent |
| Severity |  LOW SEVERITY |
| Description | Throughout the codebase, the fee denominator is sometimes set to 10_000 and sometimes to 1_000_000. This creates a bad user experience because it will be harder for users and projects to know what the fee for a pool is. |
| Recommendation | Consider updating the denominators to be the same across the contracts. This will make the contracts more easy to read as no conversions between contracts would need to be done. |
| Resolution |  RESOLVED |



2.2 BrewlabsERC20

BrewlabsERC20 is a contract that defines a skeleton for an ERC20 token that is implemented by BrewlabsPair.



2.2.1 Issues & Recommendations

| Issue #02 | Typographical issues |
|----------------|--|
| Severity |  INFORMATIONAL |
| Description | <p>Various functions do not implement safeguards for the address(0) check:</p> <ul style="list-style-type: none">- The approve function should check that the owner or spender is not address(0).- The transfer function should check that from or to are not address(0). |
| Recommendation | Consider fixing the typographical issues. |
| Resolution |  RESOLVED |

| Issue #03 | Lack of events for approval update within transferFrom |
|----------------|--|
| Severity |  INFORMATIONAL |
| Description | <p>Functions that affect the status of sensitive variables should emit events as notifications.</p> <p>transferFrom might update the approval if the user approved less than the max uint256. However, no events will be emitted for this update which can cause some discrepancies between off-chain and on-chain data.</p> |
| Recommendation | Consider emitting an event in the transferFrom function when the approval is updated. |
| Resolution |  RESOLVED |

2.3 BrewlabsFactory



BrewlabsFactory is a contract that handles the permissionless deployment of pairs for specific tokens. It uses the CREATE2 opcode that deterministically deploys to a specific address based on the bytecode of the pair contract and a salt. The salt comprises the token addresses — in this way, it ensures that the deploys are deterministic and unique for any 2 tokens.



2.3.1 Privileged Functions

- `transferOwnership`
- `setFeePercentOwner`
- `setSetStableOwner`
- `setFeeTo`
- `setFeeManager`
- `setDiscountManager`
- `setOwnerFee`
- `addToBlackList`
- `setWhitelist`



2.3.2 Issues & Recommendations

| Issue #04 Blacklist mechanism can be circumvented | |
|---|---|
| Severity |  LOW SEVERITY |
| Description | <p>The BrewLabs team adopted a blacklist approach to blacklist specific addresses for swaps. While in theory this would work, we must point out that the blacklist mechanism works only for addresses that are known, which are usually known smart contracts (e.g. Uniswap).</p> <p>For an EOA or any other address that someone can easily change, the blacklist mechanism can be easily bypassed by just using another address that calls the method where <code>isAllowedToSwap</code> is called.</p> |
| Recommendation | <p>The issue does not have an easy fix as the opposite, making the swaps only working with a whitelist approach would create an undesirable user experience as every swapper must be approved beforehand.</p> <p>We mention this so it can be documented accordingly.</p> |
| Resolution |  ACKNOWLEDGED |
| | <p>The team has stated that the function is a precautionary measure and not a complete solution to protect against MEV sandwich attacks.</p> |

| | |
|-----------------------|--|
| Issue #05 | feeTo cannot be set to address(0) |
| Severity |  LOW SEVERITY |
| Description | <p>The feeTo field within the factory is used by the pair to determine if a fee is applied on swap or not. The determination is done by comparing the feeTo variable to address(0).</p> <p>Within the factory, the owner of the factory can set this field to a specific address, except address(0).</p> <p>This limits the possibility of disabling the fees if needed.</p> |
| Recommendation | Consider if this is the desired behavior, if not, consider allowing the factory owner to set this field to address(0). |
| Resolution |  RESOLVED |

Severity

 INFORMATIONAL

Description

Line 62

* Return `true` if `address` is a type of EOA. `Other` than that it checks out

The comment is inaccurate as the `isAllowedToSwap` function does not check for EOAs; it checks only if a certain address is in the `isBlackListed` mapping.

Lines 135 - 139

/**

* @dev Updates the share of fees attributed to the owner

*

* Must only be called by owner

*/

This comment should be moved to the next line.

Line 143

`require(newOwnerFee > 0, "Brewlabs: ownerFee mustn't exceed minimum");`

The comment seems outdated as it should mention that the `ownerFee` must exceed minimum.

Recommendation

Consider fixing the typographical errors.

Resolution

 RESOLVED

2.4 BrewlabsNFTDiscountManager

BrewlabsNFTDiscountManager is a contract that handles the discounts applied to an address if that address holds a Brewlabs NFT stored under the `nftCollection` variable that gives a certain discount on swap fees.



WARNING: `nftCollection` can be changed at any point in time by the owner of this contract.

2.4.1 Privileged Functions

- `setCollection`
- `setCheckingLimit`
- `setDiscounts`
- `setDiscount`
- `transferOwnership`
- `renounceOwnership`



2.4.2 Issues & Recommendations

| | |
|-----------------------|---|
| Issue #07 | Manipulation of discounts by external parties |
| Severity |  HIGH SEVERITY |
| Description | <p>Because the discounts are based on the NFT that a user holds at the time of the transaction and there is no lock mechanism implemented within the Discount Manager, a third party can create a custom Router that can offer discounts to all its users even if they do not own an NFT.</p> <p>A third party can create a router does the following (assuming the router owns an NFT with the highest rarity that offers the best discount):</p> <ul style="list-style-type: none">- Transfers the user amount in (the amount that the user wants to swap)- Swaps the amount at a discount- Returns the amount to the user- As an extra step, it can even retain a small fee on the savings. |
| Recommendation | <p>Consider implementing a locking mechanism within the discount manager for certain periods that will make users lock their NFTs if they want to use the discounted swaps.</p> <p>Another idea is to use soulbound NFTs that cannot be traded, though this can cause issues for the holders that want to trade the NFTs on other marketplaces.</p> |
| Resolution |  PARTIALLY RESOLVED |
| | <p>The team has implemented logic to check if the user is a contract — if it is, then the discount is 0. A whitelisting mechanism has also been added to whitelist contracts that need to use the discount.</p> <p>We must state that this does not completely solve the issue because users can just exchange NFTs to receive discounts, kind of like OTC deals.</p> |

Severity

 MEDIUM SEVERITY

Description

Within the `discountOf` function, a for-loop is performed across all the NFTs owned by an address with a maximum of `checkLimit` iterations. The approach was implemented to limit the gas usage of this function. Unfortunately, this can return a false discount if the best discount NFT is at position `checkLimit + 1`, causing the user to not receive the right discount on fees when swapping assets.

The cap defined by the `checkLimit` variable can be changed at any point in time by the owner — if this limit is set to a high value, it can result in a DoS, making the swap function to revert.

Additionally, a bad actor can just send to the user NFTs prior to the swap that could alter the discount.

Recommendation

Consider allowing the user to specify the which NFT id he wants to use for the swap—in this way, the `discountOf` method will consume way less gas and avoid the unintended behavior mentioned above.

In order to solve this properly, we propose three options listed below in order of preference:


- The NFT id is sent within the data field of the swap function in the pair, then decoded within the Discount Manager to retrieve the id
- A mapping defined within the Discount Manager `mapping (address => uint256)` where the NFT id that should always be used for a discount is stored. This requires extra steps for the user:
 - A specific function within the manager that lets users set a preferred id that it can be used for all swaps
 - If the user sells the NFT, they need to update this mapping as well to not cause the swaps to revert or the functionality can just return 0, in that case.
- Add a parameter to the swap function within the pair that takes the NFT id that can be used for discounts. This approach can be problematic if external aggregators of multiple dexes want to use the Brewlabs pair as it has to build specific logic for this change in swap signature.

Resolution

ACKNOWLEDGED

The team has acknowledged that the first 30 NFTs a user holds is desired and will not be resolved. The team has stated that a function is present in the NFT contract that limits how many NFTs a user can hold. They plan to amend the wallet condition to hold a maximum of 30 NFTs per wallet to align with the swap discount query.



Issue #09**The order of the discounts is not enforced****Severity** LOW SEVERITY**Description**

The discounts are set using `setDiscounts`. However, the order is not checked, meaning that a higher rarity could have a lower discount than a lower one.

Recommendation

Consider enforcing that the discounts are in ascending order. This could be done following this code snippet:

```
uint256 lastDiscount;
for (uint256 i = 0; i < _discounts.length; i++) {
    require(_discounts[i] <= MAX_DISCOUNT, "Discount cannot
    exceed limit");
    require(_discounts[i] >= lastDiscount,
    "order");
    discounts[i] = _discounts[i];
    lastDiscount = _discounts[i];
}
```

Additionally, if the rarity goes beyond `discountLength`, the discount returned will be 0. We think that it should probably return the maximum discount. If so, consider updating Line 44 to:

```
uint256 _discountLength = discountLength;
return maxRarity > 0 && _discountLength > 0
    ? rarity > _discountLength
      ? _discounts[_discountLength - 1]
      : _discounts[rarity - 1]
    : 0;
```

Resolution RESOLVED

Issue #10**Gas optimizations****Severity** INFORMATIONAL**Description**Lines 37 - 38

```
for (uint256 i = 0; i < balance; i++) {  
    if (i >= checkLimit) break;
```

The if check is an expensive operation that could be completely avoided. checkLimit should also be cached.

Consider updating the for loop to:

```
uint256 _checkLimit = checkLimit;  
uint256 length = balance < _checkLimit ? balance :  
_checkLimit;  
for (uint256 i = 0; i < length; i++) {
```

Line 59

```
for (uint256 i = 0; i < _discounts.length; i++)
```

Consider caching _discounts.length to save gas in the 2 for-loops.

Consider removing the default initialization of the newly defined variables to save gas.



e.g. uint256 i = 0;

Consider using ++i instead of i++ to save some gas.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution PARTIALLY RESOLVED

| | |
|----------------|--|
| Issue #11 | Typographical issues |
| Severity |  INFORMATIONAL |
| Description | <p><u>Line 5</u></p> <pre>import {IERC721, IERC721Enumerable} from "@openzeppelin/ contracts/token/ERC721/extensions/IERC721Enumerable.sol";</pre> <p>This import can be removed.</p> <p>——</p> <p>The FEE_DENOMINATOR variable is unused, consider removing it.</p> |
| Recommendation | Consider fixing the typographical issues. |
| Resolution |  RESOLVED |



2.5 BrewlabsPair


BrewlabsPair is a contract that is forked from Uniswap LP version 2 pair with custom logic on how the fees are handled. The custom logic is as follows:

- BrewLabs has an NFT collection allows users to receive a discount on swaps
- The fees can be claimed or compounded for a specific pair if the pair has a fee manager assigned to it. In the case of compounded fees, the pair functions as a normal Uniswap pair; if the fee manager is enabled, the LPs fees are moved into a claiming mechanism where they are sent to the fee manager and an LP can claim a portion of it.
- The pair also contains a staking feature — if staking is enabled, then a portion of the fees is sent to the staking contract.
- The pair contains a stableSwap functionality — if that is enabled, then no fees are minted in the `_mintFee` function.

2.5.1 Privileged Functions

- `swap` (only non-blacklisted within the factory)
- `setFeePercent` (only `feePercentOwner` within the factory or the fee manager)
- `setStableSwap` (only `stableOwner` within the factory or the fee manager)
- `setStakingPool` (only factory's owner or the fee manager)
- `setPairTypeImmutable` (only factory's owner)
- `rescueWrongToken` (only factory's owner)

2.5.2 Issues & Recommendations

| | |
|-------------|---|
| Issue #12 | Owner fee can be set to 100% |
| Severity |  GOVERNANCE |
| Description | <p>The current design contains five types of fees:</p> <ul style="list-style-type: none">- LP Fee- Token Owner Fee- Referral Fee- Brewlabs Fee- Staking Fee <p>The LP, token owner and referral fees are distributed via the Fee Manager while the Brewlabs and Staking fee are distributed within the pair. The staking fee is sent directly to the staking contract, and if set, the Brewlabs fee is compounded and minted via the <code>_mintFee</code> method that is called when adding or removing liquidity.</p> <p>These fees are distributed from the total fee that is taken during swaps as they represent percentages of the total fee. If the Fee Manager is not set, then the Brewlabs fee becomes the total fee of the swap.</p> <p>Within the factory, <code>ownerFee</code> can be set to 100% which means that if the Fee Manager is not set, then the Brewlabs fee becomes 100% and the whole swap fee will go to the team and not be distributed to the LPs.</p> <p>Governance can also update <code>FeeManager</code> to a malicious contract that would prevent all users from swapping, minting and withdrawing their LP.</p> <p>Furthermore, governance can drain a pair by changing an imbalanced pair, such as BTC/USDC, to a stable pair as it would be very profitable to swap USDC to BTC.</p> |

PoC

Let's say there is pair with 2M USDC and 100 BTC, so BTC is \$20K:

- Governance sets the BTC/USDC pair to a stable pair
- Governance swaps 500K USDC and receive 49.8 BTC that are worth 996K.
- Governance stole 496K in a pool with a TVL of \$4M.

The fact that the governance can arbitrarily change the curve type is therefore a governance risk. The way this is addressed in Curve, a protocol with variable curve gradients, is by slowly adjusting the gradient over time, instead of all at once. This is of course difficult to accomplish with the current design.

Recommendation An option to solve this issue is to set a maximum cap for the owner fee which triggers if the fee manager is not set. The same check must also be done if the fee manager is unset — e.g. the fee manager can not be unset if the owner fee goes over the maximum cap.



We also recommend that for all governance functions that the BrewLabs team intend to use, a KYC-ed multi-signature set up should be used to limit the possibility of malicious actions in case the team addresses gets hacked.

Consider also marking pools as immutable to disable the `setStableSwap` function and prevent such a governance risk.


Resolution



The default owner fee is set to 16% and cannot be set to greater than 30%. When the fee manager is set, all the fees are sent to it as expected.

| | |
|-----------------------|---|
| Issue #13 | The remaining fee calculation rounds down unnecessarily |
| Severity |  LOW SEVERITY |
| Location | <u>Lines 267-270</u> <pre>uint256 remainingFee0 = amount0In * constraint.realFee / FEE_DENOMINATOR * constraint.remainingFee / constraint.operationFee; uint256 remainingFee1 = amount1In * constraint.realFee / FEE_DENOMINATOR * constraint.remainingFee / constraint.operationFee;</pre> |
| Description | The remaining fee calculation performs a division before multiplication, which rounds down the result. |
| Recommendation | Consider performing multiplications before divisions: <pre>uint256 remainingFee0 = amount0In * constraint.realFee * constraint.remainingFee / (FEE_DENOMINATOR * constraint.operationFee); uint256 remainingFee1 = amount1In * constraint.realFee * constraint.remainingFee / (FEE_DENOMINATOR * constraint.operationFee);</pre> |
| Resolution |  RESOLVED |



Issue #14**Read-only reentrancy via getReserves on the FeeManager****Severity** LOW SEVERITY**Description**



Throughout the contract, a callback to the IBrewlabsSwapFeeManager is done before updating the reserves by calling update.



The FeeManager does not use getReserves which calls reserves that have not yet been updated, but the Brewlabs team should be aware of this read-only reentrancy risk because the FeeManager can be replaced by another version.

Recommendation

Consider documenting this clearly and take necessary precautions in case FeeManager will ever use getReservers within the functions called in the pair. If this happens, consider calling sync before using the getReservers function.

Resolution RESOLVED

| Issue #15 | Typographical issues |
|----------------|---|
| Severity |  INFORMATIONAL |
| Description | <p><u>L72</u></p> <pre>event ReoverWrongToken(address indexed token, address to);</pre> <p>The event should be spelled as RecoverWrongToken.</p> <p><u>L121-122</u></p> <pre>price0CumulativeLast += uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed; price1CumulativeLast += uint256(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;</pre> <p>This section of the code should be allowed to overflow, so it should be in an unchecked block.</p> |
| Recommendation | Consider fixing the typographical issues. |
| Resolution |  RESOLVED |

| Issue #16 | Gas optimization |
|----------------|--|
| Severity |  INFORMATIONAL |
| Location | <p><u>L117</u></p> <pre>uint32 blockTimestamp = uint32(block.timestamp % 2 ** 32);</pre> |
| Description | The modulus is redundant and is unnecessary as uint32 already does it. Consider removing the unnecessary modulus to save some gas. |
| Recommendation | Consider implementing the gas optimization mentioned above. |
| Resolution |  ACKNOWLEDGED |

2.6 BrewlabsRouter

BrewlabsRouter is a contract that is based on the Uniswap V2 router with extra functionality within the swap functions which checks if the caller is whitelisted within the factory.

2.6.1 Privileged Functions

- `transferOwnership`
- `swapExactTokensForTokens` (only whitelisted wallet)
- `swapTokensForExactTokens` (only whitelisted wallet)
- `swapExactETHForTokens` (only whitelisted wallet)
- `swapTokensForExactETH` (only whitelisted wallet)
- `swapExactTokensForETH` (only whitelisted wallet)
- `swapETHForExactTokens` (only whitelisted wallet)
- `swapExactTokensForTokensSupportingFeeOnTransferTokens` (only whitelisted wallet)
- `swapExactETHForTokensSupportingFeeOnTransferTokens` (only whitelisted wallet)
- `swapExactTokensForETHSupportingFeeOnTransferTokens` (only whitelisted wallet)

2.6.2 Issues & Recommendations

No issues found.

2.7 BrewlabsSwapFeeManager

BrewlabsSwapFeeManager is a contract that will deal with the fee management of a Pair, when it is set. A pair would work like a normal Uniswap V2 pair if no fee manager is set. If the FeeManager is set, then the fee computation is split into 2 flows:

- One flow is managed within the pair, the staking fee and owner fee would be minted/forwarded to specific targets within the pair.
- The second flow is managed within the FeeManager which handles the LP Fee, TokenOwnerFee and ReferralFee.

These three fees will be distributed via a claim mechanism, meaning that they will be stored within the FeeManager and claimed by authorized parties. This adds a benefit to the LP providers as they will not need to unwind their LP positions in order to claim the fees.






2.7.1 Privileged Functions

- `setServiceInfo` (treasury or owner)
- `setTokenOwner` (owner)
- `setReferrer` (owner)
- `setFeeDistribution` (owner)
- `setFeeDistributionFromTeam` (owner of one of the tokens in a pair or tokenOwner of a pair)
- `renounceFeeSettingRole` (owner)
- `setStakingPool` (owner of one of the tokens in a pair or tokenOwner of a pair)
- `setStableSwap` (owner)
- `createPool` (factory)
- `lpMinted` (pair)
- `lpBurned` (pair)
- `lpTransferred` (pair)
- `notifyRewardAmount` (pair)
- `rescueWrongToken` (owner)



2.7.2 Issues & Recommendations

| | |
|----------------|---|
| Issue #17 | Governance can steal unclaimed rewards and DOS claims |
| Severity |  GOVERNANCE |
| Description | <p>rescueWrongToken can send the entire balance of the contract of any token. This can be used to steal the unclaimed rewards of the fee manager.</p> <p>The claimFee variable can be set to any value, even values higher than 100%. This could be used to steal unclaimed rewards from the FeeManager but also block users from claiming rewards if the fee is set very high as the ERC20 transfer would revert.</p> <p>Additionally, the fees can be set up to 32% and a malicious governance could detect a swap in the mempool and increase the swap fees to steal users' tokens up to their slippage value.</p> |
| Recommendation | <p>This issue is a bit complicated to resolve, unless a mapping is added to keep track of the balance of each token as rewards and only allow the owner to take out the excess from the unclaimed rewards of the pool. This mapping would need to be updated on notifyRewardAmount and rewards are claimed.</p> <p>Consider adding a maximum value for the claim fee. This value should probably not be set to 100% to make sure the governance will not be able to steal the token in the middle of a claim from a user.</p> <p>We also recommend that the BrewLabs team use a KYC-ed multi-signature account to manage all the governance functions to limit the possibility of malicious actions in case the team addresses gets hacked.</p> |
| Resolution |  ACKNOWLEDGED |
| | <p>The team has stated that all the governance issues will be resolved by moving towards a multi-signature ownership of governance actions.</p> |

Issue #18**A malicious user can steal rewards****Severity** HIGH SEVERITY**Description**

During the withdrawal of liquidity, the user needs to send the LP token to the pair contract and then burn it. These two transactions are batched in a single transaction using a router to avoid issues.

The hooks on transfer and on burn will be called to the FeeManager to update the different variables.

To save gas, the hook on transfer to the pair does not perform any actions as it will be followed by a burn with the same address as the user that sent the LP tokens that will update the right variables.

L 407

```
if (from == address(0x0) || to == address(0x0) || to == pair) return;
```

However this is a severe error as there is nothing that prevents a malicious user from sending the LP token from wallet A, but burning it to wallet B. This will allow the user to receive the fees from wallet A, but also from wallet B. This is even worse because:

- Wallet B might not have any rewardDebt, and it could allow him to receive enormous rewards amount.
- Wallet A will never be updated so he can claim the fees now, or even wait and claim later, stealing fees from other users.

Recommendation

Consider removing the gas optimisation at L407 as it creates a huge risk. The optimisation should also be removed for address(0) as users could still use this trick to steal rewards and make the FeeManager insolvent and revert on hooks, effectively DOSing the pair. Sending to the pair has the worst impact because the user will not even lose its LP.

Resolution RESOLVED

Issue #19**notifyRewardAmount function is prone to reentrancy exploits****Severity** HIGH SEVERITY**Description**

notifyRewardAmount implements a before-after pattern to make sure it has received enough tokens to account for tokens with a fee on transfer. However, there is no reentrancy guard, meaning that a hook token can reenter during the transfer to increase the number of tokens that the contract accounts for.


The reentrancy is prevented by the pair that has a reentrancy locker, however it can be bypassed by using another pair that have the same token with a hook to reenter.

This may lead to an insolvency of the hook token as the rewards would be too big.

Recommendation

Consider adding a reentrancy locker in the notifyRewardAmount.

Resolution RESOLVED

Issue #20**If a FeeManager is set, unset and set again, rewards may be stolen****Severity** HIGH SEVERITY**Description**

If the FeeManager is ever unset and set again, a malicious user could steal the rewards following these steps:

- FeeManager is unset.
- Malicious user deposits some liquidity.
- FeeManager is set again.

However, the rewardDebt of the malicious user was not updated, so they can steal the rewards of other users by calling `claim`.

Recommendation

Consider enforcing that a FeeManager cannot be set again once it has been unset. A new FeeManager should be set in case this ever happens.

Resolution PARTIALLY RESOLVED

If the `feeManager` is ever unset, then no `feeManager` can be set again. However, a previous `feeManager` can be re-set again which would lead to the same issue.



| | |
|-----------------------|---|
| Issue #21 | Checks-effects-interactions pattern is not adhered to |
| Severity |  HIGH SEVERITY |
| Location | <p>Line 521-531</p> <pre> uint256 fee0 = pendingAmount0 * claimFee / 10000; uint256 fee1 = pendingAmount1 * claimFee / 10000; IERC20(pool.token0).safeTransfer(treasury, fee0); IERC20(pool.token0).safeTransfer(to, pendingAmount0 - fee0); IERC20(pool.token1).safeTransfer(treasury, fee1); IERC20(pool.token1).safeTransfer(to, pendingAmount1 - fee1); pool.rewards[0] = pool.rewards[0] - pendingAmount0; pool.rewards[1] = pool.rewards[1] - pendingAmount1; pool.rewardsOfLpProvider[0] = pool.rewardsOfLpProvider[0] - pendingAmount0; pool.rewardsOfLpProvider[1] = pool.rewardsOfLpProvider[1] - pendingAmount1; </pre> |
| Description | <p>The checks-effects-interactions pattern is a known pattern in Solidity smart contract development that reduces the risk of reentrancy attacks.</p> <p>Within <code>_claimLpProviderFee</code>, the fee is sent to the <code>to</code> and <code>treasury</code> then the updates on the state is made. This can create a reentrancy attack if the <code>token0</code> or <code>token1</code> within the pair has a callback on transfer, like ERC777 tokens.</p> |
| Recommendation | Consider first updating the state then do the <code>safeTransfer</code> of the tokens at the final of the function. |
| Resolution | <p> FAILED RESOLUTION</p> <p>The transfer is still done before the debt is updated.</p> <p>This issue was upgraded to <i>high</i> as it now allows the caller to steal all rewards as the transfer is done before the the debt is updated for the caller.</p> <p>Furthermore, the function is now used in the middle of <code>lpTransferred</code> — it should be moved at the end to respect CEI.</p> |

| | |
|-----------------------|--|
| Issue #22 | The rewardDebt calculation rounds down allowing users to steal some rewards |
| Severity | <div><div></div> MEDIUM SEVERITY</div> |
| Description | <p>Within the lpMinted and lpTransferred, functions the reward debt is increased or decreased from the debt of the amount transferred.</p> <p>However, this is unsafe as rewardDebt will be rounded down and users will be able to claim more rewards than they should be able to. This could lead to an exploit where the FeeManager does not have enough funds to pay the rewards of users.</p> |
| Recommendation | <p>Consider claiming the tokens on any balance update and update the rewardDebt with the full balance of the user. However, this would allow people to claim for other people (even though the rewards will still be sent to the right users). This might be an issue for your users, especially vaults and 3rd parties.</p> <p>If this an issue for the team, consider storing the rewards of users in a mapping and on claim calculate the rewards and add the stored amounts.</p> |
| Resolution | <div><div></div> PARTIALLY RESOLVED</div> |



Issue #22**The tokenOwner logic can become malicious in specific scenarios****Severity** MEDIUM SEVERITY**Description**

Within the FeeManager, a tokenOwner can be defined for a pair. The tokenOwner is the one who receives fees and can call specific governance functions. The logic for settings this is split in 2:

- The BrewLabs team can set up this manually by calling setTokenOwner which can set the variable to any address.
- The tokenOwner is set automatically to the first call to a function that can be governed by this role. This logic can be described as follows: If the caller is the owner of the first token of the pair or the caller of the second token of the pair, then this is automatically set as the tokenOwner.

The logic can be abused by a specific token owner when specific pairs are set.

Example: A user creates a token called MyToken and watches the mempool to see if a pair is created with MyToken, then immediately call one of the governed functions to set the tokenOwner as themselves.



Recommendation



This issue is a bit hard to resolve due to the fact that the LP creation is permissionless. As there is no danger at LP creation because the owner fee will be 0, the risk enters when the fee is changed via setFeeDistributionFromTeam.



An approach would be to automatically set the tokenOwner of the pair to the creator of the LP — if the creator is the owner of one of the tokens and remove the logic of automatically set the owner in the other governance functions, leaving only the ability for the FeeManager owner (BrewLabs team) to set it to a custom address.

Resolution RESOLVED

Only the owner of BrewlabsSwapFeeManager can set the tokenOwner.

| Issue #24 | The storage gap is incorrectly calculated |
|----------------|--|
| Severity |  LOW SEVERITY |
| Description | <p>The storage gap needs to be calculated using how much storage slots were used.</p> <p>https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps</p> |
| Recommendation | <p>Consider updating the gap variable of the struct pool to 28 as it uses 22 storage slots. Also consider updating the global storage gap to 46 as only 4 slots are used.</p> |
| Resolution |  RESOLVED |

| Issue #25 | Use call instead of transfer within rescueWrongToken |
|----------------|---|
| Severity |  LOW SEVERITY |
| Description | <p>rescueWrongToken is used by the owner of the contract to rescue tokens sent by mistake to the contract.</p> <p>Within this function, if the owner wants to retrieve the native token (that might be sent by mistake to the contract, even though the contract does not have a payable function at this moment, native tokens can still be sent via selfdestruct), the transfer function is used to transfer the native token out. This approach is known to be problematic for smart contract wallets like multi-signature contracts as the transfer forwards only 2100 gas to the recipient, causing most of the calls to these smart contract wallets to fail.</p> |
| Recommendation | <p>Consider replacing the transfer function with call.</p> |
| Resolution |  RESOLVED |

| | |
|-----------------------|--|
| Issue #26 | Lack of safeTransfer usage within notifyRewardAmount |
| Severity |  LOW SEVERITY |
| Location | <u>L457</u> IERC20(token).transferFrom(msg.sender, address(this), amount); |
| Description | Within notifyRewardAmount, the transfer method is used to transfer tokens from the pair to the FeeManager. This will not work for tokens that will return false on transfer (or malformed tokens that do not have a return value). |
| Recommendation | Consider using safeTransfer instead of transfer as is done throughout most of this contract. |
| Resolution |  RESOLVED |



Severity

 INFORMATIONAL

Description

Line 3

```
pragma experimental ABIEncoderV2;
```

This is obsolete as the pragma version is 0.8.14.

Line 69

```
event ReoverWrongToken(address indexed token, address to);
```

The event should be updated to RecoverWrongToken.

Line 97

```
@param pairList array of brewlabs pairs' address represent  
the pools in which keepinng fee
```

Change *keepinng* to *keeping*.

Line 208

```
* This method can called by token owner of pair
```

The comment is inaccurate as the function can be called by any owner of any of the two tokens.

Line 243

```
require(pools[pair].feeDistribution.isRenounced == false,  
"BrewlabsFeeManager: role was renounced");
```

`== false` can be replaced with `!`.

The `_claimLpProviderFee`, `_claimReferralFee` and `_claimTokenOwnerFee` should emit a custom event to track these claimed fees easily in your off-chain infrastructure.

Recommendation

Consider fixing the typographical errors.

Resolution

 PARTIALLY RESOLVED

ABIEncoderV2 is still present.

Severity

 INFORMATIONAL

Description

L218 - 223

```
if (pools[pair].tokenOwner == address(0x0)) {  
  require(_checkTokenOwner(pair), "BrewlabsFeeManager: caller  
is not token's owner");  
  pools[pair].tokenOwner = msg.sender;  
} else {  
  require(msg.sender == pools[pair].tokenOwner,  
"BrewlabsFeeManager: caller is not token owner");  
}
```

Consider caching `pools[pair].tokenOwner` to save some gas.

The Pool struct contains unnecessary writing to storage. `rewards` and `rewardsOfLpProvider` can be tracked off-chain to save a considerable amount of gas.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution

 PARTIALLY RESOLVED

`rewards` and `rewardsOfLpProvider` were not resolved.

2.8 BrewlabsLibrary

BrewlabsLibrary is a utility library that implements utility functions around the pair.

2.8.1 Issues & Recommendations

No issues found.





PALADIN
BLOCKCHAIN SECURITY