



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Trader Joe (Core v2.2)

31 March 2024



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	4
1 Overview	5
1.1 Summary	9
1.2 Contracts Assessed	10
1.3 Findings Summary	11
1.3.1 LBFactory	12
1.3.2 LBPair	12
1.3.3 LBRouter	12
1.3.4 BinHelper	12
1.3.5 OracleHelper	12
1.3.6 ReentrancyGuardUpgradeable	13
1.3.7 Uint256x256Math	13
1.3.8 Hooks	13
1.3.9 LBBaseHooks	13
1.3.10 LBHooksManager	13
1.3.11 LBHooksBaseRewarder	14
1.3.12 LBHooksRewarder	14
1.3.13 LBHooksExtraRewarder	15
1.3.14 MasterChef	15
2 Findings	16
2.1 Core/LBFactory	16
2.1.1 Privileged Functions	17
2.1.2 Issues & Recommendations	18
2.2 Core/LBPair	19
2.2.1 Privileged Functions	20
2.2.2 Issues & Recommendations	21
2.3 Core/LBRouter	22

2.3.1 Privileged Functions	22
2.3.2 Issues & Recommendations	22
2.4 Core/BinHelper	23
2.4.1 Issues & Recommendations	23
2.5 Core/OracleHelper	24
2.5.1 Issues & Recommendations	25
2.6 Core/ReentrancyGuardUpgradeable	26
2.6.1 Issues & Recommendations	26
2.7 Core/Uint256x256Math	27
2.7.1 Issues & Recommendations	27
2.8 Core/Hooks	28
2.8.1 Issues & Recommendations	28
2.9 Core/LBBaseHooks	29
2.9.1 Issues & Recommendations	30
2.10 LB-Rewarder/LBHooksManager	31
2.10.1 Privileged Functions	32
2.10.2 Issues & Recommendations	33
2.11 LB-Rewarder/LBHooksBaseRewarder	35
2.11.1 Privileged Functions	36
2.11.2 Issues & Recommendations	37
2.12 LB-Rewarder/LBHooksRewarder	53
2.12.1 Privileged Functions	53
2.12.2 Issues & Recommendations	54
2.13 LB-Rewarder/LBHooksExtraRewarder	56
2.13.1 Privileged Functions	56
2.13.2 Issues & Recommendations	57
2.14 Moe-Core/MasterChef	59
2.14.1 Issues & Recommendations	59

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 Trader Joe's Core v2.2 contracts on the Mantle network. 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.

TraderJoe Core V2.2 is built upon the previously audited V2.1. The concentrated liquidity DEX implements a unique approach which differs from UniswapV3's ranged tick concept and introduces bins, where each bin represents a specific price. Unlike UniswapV3, a swap consumes all consecutive bins without slippage. This brings about the unique scenario where a full swap can be executed with a nominal price if the bin liquidity allows for it.

The previous audit can be found here: <https://paladinsec.co/projects/trader-joe/>

What is new in Trader Joe Core V2.2?

The Trader Joe development team introduced a hook concept like that of other similar protocols. This means that upon interactions with the pair, a hook is invoked, namely:

- swap
- flashloan
- mint
- burn
- setHookParameters
- batchTransferFrom

A differential audit was carried out for the following contracts:

- LBFactory
- LBPair
- LBRouter
- BinHelper
- OracleHelper
- ReentrancyGuardUpgradeable
- Uint256x256Math
- MasterChef

A full, ground-up audit was carried out for the following contracts:

- Hooks
- LBBaseHooks
- LBHooksManager
- LBHooksBaseRewarder
- LBHooksRewarder
- LBHooksExtraRewarder

Merchant Moe LB Rewarder

The Merchant Moe LB-Rewarder repository contains rewarder logic meant to be used with the introduced hooks. All contracts within this repository are inherently connected with each other. The following steps are usually done during deployment:

1. Deploy a new `LBHooksBaseRewarder` contract. Upon deployment, a pool on the Masterchef is created for this rewarder. Afterwards, `setLBHooksParametersOnPair` within the factory is invoked to set the corresponding hook parameters on the pair. Upon setting the hook, a call to the new rewarder is then executed which initializes the contract and mints 1 nominal token and deposits it into the masterchef, setting the rewarder up for gaining rewards.
2. Deploy a new `LBHooksExtraRewarder` contract. Upon deployment, `setLBHooksExtraRewarder` is triggered on the `LBHooksBaseRewarder` contract which forges and sets the `_extraHooksParameters` which sets up `LBHooksExtraRewarder`. Afterwards, the owner can then set the reward data.

The following image is an example flow based on the `beforeSwap` hook:

1.1 Summary

Project Name	Trader Joe
URL	http://traderjoexyz.com/
Platform	Mantle
Language	Solidity
Preliminary contracts	https://github.com/traderjoe-xyz/joe-v2/tree/ab7689e063e84dc69b2aa1e74e52883590c6218e/src https://github.com/merchant-moe/lb-rewarder/blob/9a388591549871e6a9fcc6dcadd97b3e9f92e710 https://github.com/merchant-moe/moe-core/tree/11200e28e065eab7a7bf2107fb9711bf6e5750c8
Resolution #1	joe-v2: 24efde2ee26b1c65aebcc97a9ea848693b4d05d4 lb-rewarder: 1115753c2f6e4363eb55f170602c9cf9d8ed20c2 moe-core: no changes



1.2 Contracts Assessed

Name	Contract	Live Code Match
LBFactory	0xa6630671775c4EA2743840F9A5016dCf2A104054	✓ MATCH
LBPair	Implementation: 0xf6863Db7323aaC43fE8AEF0B3eF63AA6b32DDB3B	✓ MATCH
LBRouter	0xAFb85a12Babfafabfe1a518594492d5a830e782a	✓ MATCH
BinHelper	Dependency	✓ MATCH
OracleHelper	Dependency	✓ MATCH
ReentrancyGuardUpgradeable	Dependency	✓ MATCH
Uint256x256Math	Dependency	✓ MATCH
Hooks	Dependency	✓ MATCH
LBBaseHooks	Undeployed	UNDEPLOYED
LBHooksManager	Undeployed	UNDEPLOYED
LBHooksBaseRewarder	Undeployed	UNDEPLOYED
LBHooksRewarder	Undeployed	UNDEPLOYED
LBHooksExtraRewarder	Undeployed	UNDEPLOYED
MasterChef	Undeployed	UNDEPLOYED

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	1	-	-	1
● High	2	2	-	
● Medium	4	2	-	2
● Low	7	4	-	3
● Informational	12	5	-	7
Total	26	13	-	13

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 LBFactory

ID	Severity	Summary	Status
01	GOV	Governance Privilege: Hook modification privilege	ACKNOWLEDGED

1.3.2 LBPair

ID	Severity	Summary	Status
02	MEDIUM	Subsequent liquidity additions can result in overflow of overall bin liquidity	✓ RESOLVED

1.3.3 LBRouter

No issues found.

1.3.4 BinHelper

No issues found.

1.3.5 OracleHelper

ID	Severity	Summary	Status
03	INFO	Oracle is still vulnerable to manipulation	ACKNOWLEDGED

1.3.6 ReentrancyGuardUpgradeable

No issues found.

1.3.7 Uint256x256Math

No issues found.

1.3.8 Hooks

No issues found.

1.3.9 LBBaseHooks

No issues found.

1.3.10 LBHooksManager

ID	Severity	Summary	Status
04	INFO	Typographical issues	✓ RESOLVED
05	INFO	Lack of address(0) check after cloning	ACKNOWLEDGED

1.3.11 LBHooksBaseRewarder

ID	Severity	Summary	Status
06	HIGH	A malicious party can DoS the pair and hook, disabling the rewarder permanently	✓ RESOLVED
07	MEDIUM	getPendingRewards returns incorrect pending rewards	✓ RESOLVED
08	MEDIUM	Users may grief rewards by abusing known attack-vector to temporarily DoS swaps	ACKNOWLEDGED
09	MEDIUM	Contracts without fallback functions might not accept the native token	ACKNOWLEDGED
10	LOW	Unfair reward distribution due to using liquidity to determine pro-rata amount	ACKNOWLEDGED
11	LOW	Delta bins are not set upon initialization	ACKNOWLEDGED
12	LOW	_claim will revert if contract has insufficient tokens	✓ RESOLVED
13	LOW	Inconsistency in token claim for recipients	✓ RESOLVED
14	LOW	External call to extraRewarder is executed before MOE reward	✓ RESOLVED
15	LOW	Rewards are distributed retroactively if the active range has no liquidity (BaseRewarder)	ACKNOWLEDGED
16	LOW	Automatic reward claim largely increases attack surface	✓ RESOLVED
17	INFO	Unsafe casting within _updateUser function	✓ RESOLVED
18	INFO	Logic is not compatible with extraRewarder from Merchant Moe MasterChef	ACKNOWLEDGED
19	INFO	Users can only claim as long as the rewarder is linked	ACKNOWLEDGED
20	INFO	Sophisticated but unrealistic exploit allows for DoS in the pair under extreme situations	ACKNOWLEDGED
21	INFO	Typographical issues	✓ RESOLVED

1.3.12 LBHooksRewarder

ID	Severity	Summary	Status
22	INFO	Broken MasterChef prevents state changes within LBPair	ACKNOWLEDGED
23	INFO	Removal of ExtraRewarder will prevent users from claiming rewards	ACKNOWLEDGED
24	INFO	Typographical issues	✓ RESOLVED

1.3.13 LBHooksExtraRewarder

ID	Severity	Summary	Status
25	HIGH	Unconsidered edge-case within _getPendingTotalRewards can result in DoS of reward update due to underflow	✓ RESOLVED
26	INFO	Typographical issues	✓ RESOLVED

1.3.14 MasterChef

No issues found.

2 Findings

2.1 Core/LBFactory

This audit is based on the following changes: <https://www.diffchecker.com/2rnBFogs/>

The new version includes the following changes:

- Implementation of `Ownable2Step` and `AccessControl`.
- Introduction of `LB_HOOKS_MANAGER_ROLE`, which allows role owners to modify hooks for `LBPairs`.
- Functionality to add and remove hooks on `LBPairs`.
- Modification of `AccessControl` logic which allows only the owner to set roles.

Hooks can be set as follows:

`LBFactory.setLBHooksParametersOnPair`



`LBPair.setHooksParameters`



`Hooks.onHooksSet`

2.1.1 Privileged Functions

- `grantRole`
- `revokeRole`
- `transferOwnership`
- `renounceOwnership`
- `setLBPairImplementation`
- `setLBPairIgnored`
- `setPreset`
- `setPresetOpenState`
- `removePreset`
- `setFeesParameterOnPair`
- `setFeeRecipient`
- `setFlashLoanFee`
- `addQuoteAsset`
- `removeQuoteAsset`
- `forceDecay`



2.1.2 Issues & Recommendations

Issue #01	Governance Privilege: Hook modification privilege
Severity	GOVERNANCE
Description	The contract owner has full control over the LB_HOOKS_MANAGER_ROLE and can assign this role to any arbitrary address. Any address with this role can effectively brick any pair completely by adding a malicious hook.
Recommendation	Consider incorporating a Gnosis multi-signature contract as the owner and ensuring that the participants are trusted entities. The same should be applied for any address with the LB_HOOKS_MANAGER_ROLE.
Resolution	ACKNOWLEDGED The team stated: "The owner will be our main multisig. The LB_HOOKS_MANAGER_ROLE will be given only to our multisig and to the hooks manager contract."

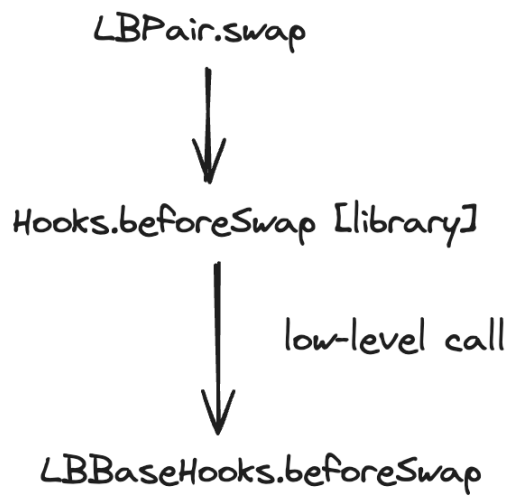
2.2 Core/LBPair

This audit is based on the following changes: <https://www.diffchecker.com/pp3RMs6V/>

The key updates to the LBPair contract are as follows:

- Inheriting `ReentrancyGuardUpgradeable` instead of `ReentrancyGuard`. This allows different scenarios to be customized and was specifically implemented for hooks.
- Flashloan fees are now solely redistributed to the protocol instead to the LP pair — this significantly reduces the attack surface vector and Paladin very much appreciates this change.
- Introduction of hooks:
 - before/after swap
 - before/after flashloan
 - before/after mint
 - before/after burn
 - before/after batchTransferFrom
- Implementation of `setHooksParameters` to allow the factory owner to change the hook for this LBPair.

Hooks are invoked as follows:



2.2.1 Privileged Functions

- `initialize`
- `setStaticFeeParameters`
- `setHooksParameters`
- `forceDecay`
- `collectProtocolFees`

2.2.2 Issues & Recommendations

Issue #02	Subsequent liquidity additions can result in overflow of overall bin liquidity
Severity	 MEDIUM SEVERITY
Description	There is no check to ensure that the overall liquidity for a bin is larger than uint256 after liquidity has been added. The only checks done are before adding liquidity for the bin liquidity and the liquidity to be added. Therefore, swap fee accumulation and also the adding of liquidity can trigger the bin liquidity to become larger than uint256. This will prevent users from further adding liquidity and will result in down-stream issues for rewarders.
Recommendation	Consider implementing a proper limitation that ensures the bin liquidity can never overflow. While this is trivial for the liquidity addition itself, it can become more tricky for fee accumulation.
Resolution	 RESOLVED

2.3 Core/LBRouter

This audit is based on the following changes: <https://www.diffchecker.com/DWo8DJ7O/>

No significant changes were made, only some slight adjustments and one bug fix.

2.3.1 Privileged Functions

- sweep
- sweepLBToken

2.3.2 Issues & Recommendations

No issues found.



2.4 Core/BinHelper

This audit is based on the following changes: <https://www.diffchecker.com/HClYbCj3/>

The only notable change is the adjustment of the initial supply to be the square root of userLiquidity.

2.4.1 Issues & Recommendations

No issues found.



2.5 Core/OracleHelper

This audit is based on the following changes: <https://www.diffchecker.com/WBlHkLuo/>

A slight change has been made to make the TWAP oracle more secure. The starting `activeID` will now be used instead of the `activeID` post-swap to make it more robust against potential manipulation. It goes without saying that TWAP oracles are always to be used with caution since they are inherently vulnerable to manipulation.



2.5.1 Issues & Recommendations

Issue #03	Oracle is still vulnerable to manipulation
Severity	 INFORMATIONAL
Description	Since samples are only updated once a threshold time after the last update has passed, a malicious user could push the price to a desired value (malicious bin), wait one block and execute a swap back. As the oracle uses activeID at the start of the swap, the malicious bin will be used. This issue is only rated as informational due to the following reasons: a) The attacker needs to be able to sort transactions in blocks to ensure that he is the last swapper in the recent block and the first swapper in the next block to ensure no funds are lost. b) TWAP oracles are inherently vulnerable to manipulation — this is a widely acknowledged fact.
Recommendation	If the team wishes to make changes, consider decreasing the sampleLifeTime to incorporate more data.
Resolution	 ACKNOWLEDGED

2.6 Core/ReentrancyGuardUpgradeable

ReentrancyGuardUpgradeable is a simple fork of OpenZeppelin's ReentrancyGuardUpgradeable contract (<https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/utils/ReentrancyGuardUpgradeable.sol>)

The only change is that the visibility for `_nonReentrantBefore` and `_nonReentrantAfter` has been changed from `private` to `internal` to implement them directly in child contracts.

2.6.1 Issues & Recommendations

No issues found.



2.7 Core/Uint256x256Math

This audit is based on the following changes: <https://www.diffchecker.com/4HrupKVF/>

This contract was extended with a function to include a sqrt calculation, which is solely used for minting the initial LBToken.

2.7.1 Issues & Recommendations

No issues found.



2.8 Core/Hooks

The Hooks library is responsible for the correct encoding and decoding of hook parameters. It essentially encodes the hooks address and the different boolean flags, such as `beforeSwap` and `afterSwap` into a single `bytes32` variable, and the bit positions from 0-159 represent the hook address and the bit positions from 160 to 169 the hook settings.

Additionally, it is called by specific functions — if, for example, the `beforeSwap` flag is set to `true`, the `beforeSwap` function will be invoked and execute a low-level call to the target hooks address, with `beforeSwap` expected as return data.

In the current protocol state, the target contract is `LBaseHooks`, which then will be further extended with rewarder logic.

A full audit was carried out for this contract.

2.8.1 Issues & Recommendations

No issues found.



2.9 Core/LBBaseHooks

LBBaseHooks forms the base contract for the hook implementation and is an abstract contract meant to be inherited by other logic contracts. This contract is invoked by the Hooks library (either from the pair or from the parent rewarder) and is meant to be the "hooks" address. As a base contract, it only includes external functions which ensure the caller is only the corresponding LP pair (base)/ parent rewarder (extra).

The internal functions do not have any logic and are meant to be overridden. The LBBaseHooks contract is the main parent contract of extra and base rewarders and employs the standard interface and takes care of the correct access control.

Trader Joe has developed the following hook strategies:

1. **LBHooksManager**: This is the factory contract for deploying standard and extra rewarders.
2. **LBHooksBaseRewarder**: This contract incorporates the main rewarder logic which updates the accumulator and allows users to claim their rewards. This contract is meant to be the parent contract for the standard and extra rewarder contracts and inherits the LBBaseHooks contract.
3. **LBHooksRewarder**: This contract serves as a child contract for the LBHooksBaseRewarder, employing the base reward logic. It receives reward tokens from a corresponding masterchef position.
4. **LBHooksExtraRewarder**: This contract is the equivalent to the LBHooksRewarder with the only difference being that rewards are not dependent on masterchef interactions but can be freely set depending on the contract balance. Reward tokens must be sent directly to this contract. It can be only applied on top of the base rewarder.

A full audit was conducted for this contract.

2.9.1 Issues & Recommendations

No issues found.

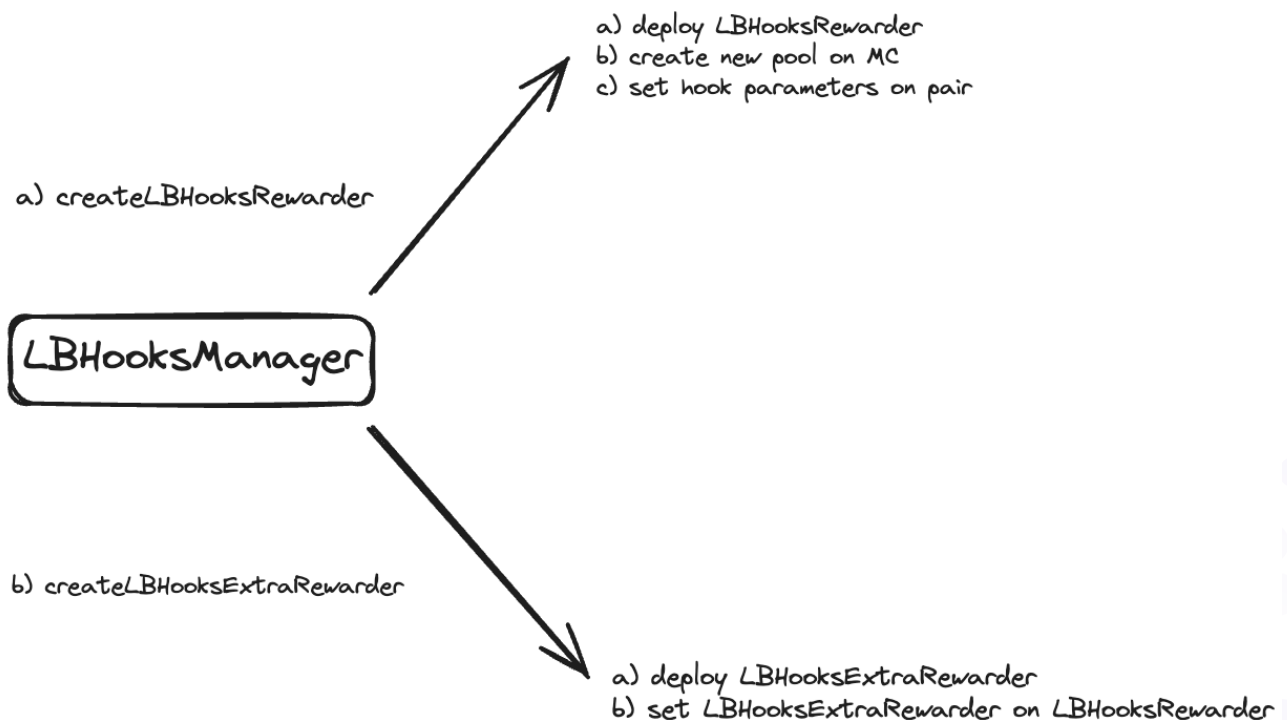
2.10 LB-Rewarder/LBHooksManager

LBHooksManager is the factory contract for deploying the HooksRewarder and HooksExtraRewarder contracts. It is also meant to be the implementation for a proxy contract, and deploys both contracts as minimalistic proxy contracts using Trader Joe's ImmutableClone library.

Upon deployment of the BaseRewarder, a pool within the MasterChef is created and the BaseRewarder is linked to the LBPair.

Upon deployment of the ExtraRewarder, the ExtraRewarder is linked to the BaseRewarder.

The deployment and setup process works as follows:



The contract owner can determine preset hook parameters for both types: Rewarder and ExtraRewarder. These presets will then be used on the corresponding type deployment.

2.10.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `setLBHooksParameters`
- `createLBHooksRewarder`
- `createLBHooksExtraRewarder`



2.10.2 Issues & Recommendations

Issue #04	Typographical issues
Severity	INFORMATIONAL
Description	<u>L28</u> IERC20 internal immutable _moe; This variable is unused. Consider removing it.
Recommendation	Consider fixing the typographical issue(s).
Resolution	RESOLVED

Issue #05**Lack of address(0) check after cloning****Severity** INFORMATIONAL**Description**

Once the proxy has been deployed, the code does not check if the deployment was successful:

```
rewarder =  
ILBHooksRewarder(_cloneHooks(LBHooksType.Rewarder,  
Hooks.getHooks(hooksParameters), immutableData));
```

Recommendation

Consider ensuring that `rewarder != address(0)`. The same should be considered for `extraRewarder`.

Resolution ACKNOWLEDGED

The team stated that the library already checks for failed deployment.

[ImmutableClone:133](#)

```
instance := create2(0, sub(data, 0x1f), add(extraLength,  
0x3f), salt)
```

```
// If `instance` is zero, revert.  
if iszero(instance) {  
    // Store the function selector of `DeploymentFailed()`.  
    mstore(0x00, 0x30116425)  
    // Revert with (offset, size).  
    revert(0x1c, 0x04)  
}
```

2.11 LB-Rewarder/LBHooksBaseRewarder

LBHooksBaseRewarder is the core contract of the reward system. It uses a similar process as the Synthetix contract to distribute rewards, where rewards are distributed to users based on their balance of IDs.

The contract implements a specific reward range which is always dependent on the current `activeID`, and the owner can set `deltaBinA` and `deltaBinB` which are used to determine the ranges based on the `activeID`. While these can be any arbitrary value (as long as they do not have a delta larger than `MAX_NUBER_OF_BINS`), it is expected that they form a fair range around the `activeID` – for example, `deltaBinA = 5` and `deltaBinB = -5`.

On each state-changing transaction, the range around the current `activeID` is fetched (which was active since the last state-changing transaction) and rewards are fairly distributed amongst these IDs. In detail, before each state change, rewards are allocated as follows:

- a) Determine the range to be rewarded, which is usually -5 bins and +5 bins around the `activeID`.
- b) Aggregate the total liquidity from all IDs in the rewarded range and determine the pro-rata weight from each ID on the total liquidity.
- c) Increase `accRewardsPerShare` (accumulator) for each ID by the weight percentage and the total pending rewards (since the last update), divided by the `totalSupply` of each specific bin to represent how much rewards are assigned to each nominal token.

Whenever users transfer tokens, mint or burn liquidity, they will receive rewards based on their current balance and the delta of `accRewardsPerShare`. These rewards are stored in the `_unclaimedRewards` mapping and transferred out after

mints, burns and token transfers. Additionally, users can also simply claim these rewards.

The Base is designed to execute logic on the following hooks:

- BEFORE_SWAP
- BEFORE_MINT
- AFTER_MINT
- BEFORE_BURN
- AFTER_BURN
- BEFORE_TRANSFER
- AFTER_TRANSFER

This contract is used as an abstract contract only and is inherited by the LBHooksRewarder and LBHooksExtraRewarder contracts.

A full audit was conducted on this contract.

2.11.1 Privileged Functions

- transferOwnership
- renounceOwnership
- setDeltaBins
- sweep

2.11.2 Issues & Recommendations

Issue #06	A malicious party can DoS the pair and hook, disabling the rewarder permanently
Severity	 HIGH SEVERITY
Description	The reward mechanism relies on the correctness of the aggregated liquidity calculation amongst all IDs in the rewarded range: <pre>for (uint256 i; i < length; ++i) { uint24 id = uint24(ids[i]); (uint128 binReserveX, uint128 binReserveY) = lbPair.getBin(id); uint256 totalSupplyX64 = lbPair.totalSupply(id); uint256 liquidityX128 = BinHelper.getLiquidity(binReserveX, binReserveY, activePriceX128); liquiditiesX128[i] = liquidityX128; totalSuppliesX64[i] = totalSupplyX64; totalLiquiditiesX128 += liquidityX128; }</pre> Liquidity is calculated as follows: $L = x * P + y$ The price P is determined by the current activeID. For example, with an activeID of 9_275_880, the price is <pre>115792089237316195423570985008687907853269984665640564039 457584007913129639935</pre> While naturally upon minting it is ensured that the liquidity amount itself will not overflow, the addition of two subsequent IDs liquidity can overflow. An attacker can trivially exploit this and DoS the protocol: 1. Swap towards a very distant bin — ideally the upper boundary (the higher the ID, the less tokenX is needed).

-
2. Supply tokenX around this ID, each of their own ID's liquidity will be $< \text{uint256}$, however, in addition they will be $> \text{uint256}$. (It would already be sufficient to supply 500000000000 nominal tokenX to 9_275_880 and 9_275_879 to cause the overflow, for instance, this would be 0,0000005 WAVAX).
 3. Upon the next state change and call of `_updateAccruedRewardsPerShare`, the call will revert due to an overflow. The attacker has successfully DoSed the pair and hook.

While unplugging will re-enable state transitions for the pair itself, re-plugging to a new rewarder will not work. This effectively causes a DoS to the full reward system completely and permanently.

It is notable that due to a flaw within the core, it is possible to add overall liquidity with an amount $> \text{uint256}$ to a bin. This will also result in an overflow of the `getLiquidity` function and a DoS.

Recommendation	Consider modifying the core in such a way that liquidity amongst a certain range is $< \text{uint256}$. Such a change in the core would need intrusive additional validation to ensure it does not have undesired side-effects. It is important to mention that this check would only be valid if the <code>deltaBins</code> are on both sides. If for example <code>deltaBinA = 100</code> and <code>deltaBinB = 110</code> , such a check would be insufficient.
-----------------------	---

Resolution



The team stated: "The core has been modified so that bin's liquidity is always lower than `MAX_UINT256 / maxNbOfBin`"

Issue #07**getPendingRewards returns incorrect pending rewards****Severity** MEDIUM SEVERITY**Description**

getPendingRewards solely relies on the pending rewards within the masterchef, as can be seen in the `_getPendingTotalRewards` function within `LBHooksRewarder`.

The problem is if there are tokens transferred directly to the contract or the aggregated liquidity was zero during the last update, there will be leftover tokens which are distributed in the next update.

Additionally, if the reward token is has fees on transfer, the fees are not reflected.

This issue has been marked as medium severity despite it being a view-only function as third-party protocols will rely on it.

Recommendation

Consider following the same approach as done in the `_updateRewards` function and aggregating any remaining token balance. To handle reward tokens with a fee on transfer, there is no simple solution.

Resolution RESOLVED

The team stated: "Any remaining reward token is now added to the `pendingTotalRewards` for the main rewarders; the reward token from the masterchef will only be MOE which does not have a fee on transfer."

Issue #08**Users may grief rewards by abusing known attack-vector to temporarily DoS swaps****Severity** MEDIUM SEVERITY**Description**

Since rewards are distributed around the activeID, this naturally opens up the possibility to add liquidity to a very unconventional bin and swap towards it. As rewards are based on the passed timestamp since the last update, the longer the pair stays in that position, the longer the user will receive rewards.

For CLAMMs, there are inherently a few design issues which can make it more difficult to execute swaps, which are:

- a) Frontrunning swaps such that the slippage check reverts.
- b) Adding 1 wei of liquidity to a large amount of bins so that swaps become very gas-extensive or even revert due to the block gas limit being exceeded.

These scenarios are already known and acknowledged. However, with the addition of rewarders, the severity changes because there is now an economic incentive for malicious users.

Of the above scenarios, b) would make the most sense since this would most probably need governance intervention (no user would pay a large amount of gas to swap on AVAX), therefore governance would need to bear the high gas costs for such a scenario. Since governance is most probably not around 24/7, this could open a window for exploitation where no swaps would occur and the user would gain a large amount of tokens.

If we examine the potential reward amount distributed for one pool, this attack could very much make economical sense for the attacker.

Recommendation

There is no simple solution for this issue as it is inherently existent for all CLAMMs. However, we highly recommend the team to observe all rewarded pairs for stale prices.

Resolution ACKNOWLEDGED

Issue #09**Contracts without fallback functions might not accept the native token****Severity** MEDIUM SEVERITY**Description**

Reward transfers are handled within the `_safeTransfer` function, which either executes a low-level call padded with value or a simple token transfer.

The low-level call will not work for contracts without a fallback function. Specifically since there will be many vaults building on top of Trader Joe's DEX V2.2, this can pose a serious problem if those vaults are not built to support the native gas token.

Recommendation

Consider wrapping the token in case of a call revert and transferring it out regularly.

Resolution ACKNOWLEDGED

The team stated: "We don't want to introduce edge cases where a different token than the one returned by `getRewardToken` could be sent. However, the ERC20 and native handling was moved in a separate library to make it easier for protocols that build on top of the rewarders."



Issue #10**Unfair reward distribution due to using liquidity to determine pro-rata amount****Severity** LOW SEVERITY**Description**

Rewards are distributed/allocated on each state changing event such as mint, burn, swap and transfer. The way the allocation is done is based on the aggregated liquidity amongst the reward range and then distributed pro-rata towards each bin on the bins liquidity share amongst the aggregated liquidity.

The problem is that bins have different prices and the liquidity derived from tokenX is based on the current bin's price.

For example, if we have a WETH/USDC pair and at delta +1 tick, WETH is worth 4000 USDC which has a supply of 10 ETH, this will be 40_000 L.

However, if we have 4 ETH at delta +5 tick (assume 100 BPS binstep to demonstrate the issue), one ETH is worth 4204 USDC, therefore the 10 ETH will be 42_040 L.

In fact, both 4 ETH are worth the current market price.

Recommendation

Consider if this can become an issue, if yes, consider using the average price of all active bins. However, we are not of the opinion that a change is necessary and are fine with the design being kept as-is to avoid any additional undesired side-effects due to the introduction of a fix.

Resolution ACKNOWLEDGED

The team stated: "Seems outdated as the activePrice is used."

Issue #11**Delta bins are not set upon initialization****Severity** LOW SEVERITY**Description**

Whenever a base rewarder is deployed, rewards are immediately accumulated due to the deposit in the masterchef. This is not the case for the extra rewarder as it first needs the manual setting for the reward parameters.

What this means is that rewards are immediately distributed for the base rewarder. The problem is that the delta bins are not set upon initialization, which means that only the activeID will get rewards until the configuration is finalized.

Recommendation

Consider if this will become an issue, if yes, consider setting the delta bins upon the `_onHooksSet` call.

Resolution ACKNOWLEDGED

The team stated: "Seems incorrect as the `deltaBinB` is exclusive -> `[activeId + deltaBinA, activeId + deltaBinB]`, so 0,0 won't reward anything. Added a comment on the set function for more clarity."



Issue #12**_claim will revert if contract has insufficient tokens****Severity** LOW SEVERITY**Description**

_claim attempts to transfer out any rewards which are allocated to the user:

```
_claim(user, _unclaimedRewards[user])
```

```
function _claim(address user, uint256 rewards) internal  
virtual {  
    if (rewards == 0) return;  
  
    _totalUnclaimedRewards -= rewards;  
    _unclaimedRewards[user] -= rewards;  
  
    _safeTransfer(_getRewardToken(), user, rewards);  
}
```

If the contract does not have sufficient tokens to pay the user, this will result in a revert, effectively preventing mint, burn and transfer.

This issue is only rated as low severity because the corresponding rewarders take care of it. If however the BaseRewarder is deployed with other rewarders, this can pose a problem.

Recommendation

The isolated fix for this issue would be to only reduce the state variables by the amount which was transferred and implement a condition check which only transfers the contract balance if the unclaimed rewards are larger. However, as already discussed with the developer, we are of the opinion to remove the _claim on most occasions and only make it externally callable for users.

Resolution RESOLVED

The team stated: "Removed passive claiming, rewards can only be claimed when the claim function is explicitly called."

Issue #13**Inconsistency in token claim for recipients****Severity** LOW SEVERITY**Description**

Whenever tokens are transferred, `_claim` is only invoked for the sender and not for the recipient. We are of the opinion that the developer had the intention to prevent claiming for addresses that are not executors. This is a standard behavior for tax-purposes to prevent undesired tax implications. However, for the mint call, it is possible to mint to a recipient address which is not the executor. Upon `_afterMint`, `_claim` is then invoked for the recipient. This is an inconsistency for the above described logic.

Recommendation

Consider not claiming for the recipient upon `_afterMint`. This will not have any side-effects since users can still claim manually.

Resolution RESOLVED

The team stated: "Removed passive claiming, rewards can only be claimed when the `claim` function is explicitly called."



Issue #14**External call to extraRewarder is executed before MOE reward****Severity** LOW SEVERITY**Description**

The flow for claiming with an additional extraRewarder is as follows:

1. User calls `claim`.
2. `_onClaim` is triggered which claims tokens from ExtraRewarder (this can be any arbitrary token including such tokens that allow for reentrancy or the native gas token).
3. Afterwards, the `_claim` function in the base contract is invoked.

The problem stems from executing the call to ExtraRewarder before the `_claim` call — in such a scenario, a user could reenter into the `claim` function and trigger `_claim` twice. Fortunately, `_claim` does not reset the balance but instead does a deduction: `_unclaimedRewards[user] -= rewards;`

Therefore, a reentrancy attack is not possible.

However, we still think it is a good idea to do the less risky call first.

Recommendation

Consider first executing the original claim and then invoking the ExtraRewarder. Moreover, consider implementing a reentrancy guard.

Resolution RESOLVED

The team stated: "Updated the `_claim` function to follow CEI; however the transfer is kept after the `onclaim` call as it's an interaction."

Issue #15**Rewards are distributed retroactively if the active range has no liquidity (BaseRewarder)****Severity** LOW SEVERITY**Description**

During the normal business logic, there will mostly be liquidity within the rewarded range, at least within the activeID. However, if liquidity is fully withdrawn from the activeID and no swap has been done yet, it might be possible that there is no liquidity at all within the rewarded range. In such a scenario, the accumulator is not increased but the reward update rather returns early. This will just harvest from the masterchef and leave the balance in the contract to be distributed whenever there is liquidity in the rewarded range, effectively distributing these rewards retroactively.

Contrary to that, the ExtraRewarder handles this differently — in such a scenario, `lastUpdateTime` is updated but no rewards are distributed, which means rewards will simply stay in the contract and then will be allocated during the next reward data update.

Recommendation

Consider if that scenario is desired - we agree that it might be a design choice. However, if it is not desired, consider simply aggregating these rewards in a new storage variable, making it claimable for the contract owner.

Resolution ACKNOWLEDGED

Severity

 LOW SEVERITY

Description

Rewards are automatically claimed and transferred out on the following scenarios:

- afterMint
- afterBurn
- afterBatchTransferFrom

These claims do not only transfer the MOE token to the recipient but also the reward token from the ExtraRewarder, which can be the native gas token. Such a transfer opens up probable attack-surfaces for reentrancy attacks.

Most of the time, exploits happen due to arbitrary user inputs or users invoking functions which are not meant to be invoked by users — one can argue that a large user flexibility is a great seed for exploits. Therefore, at Paladin, we are of the opinion that codebases should never provide more user flexibility than necessary during the normal business logic.

This applies to automatic claims and therefore we are of the opinion to only allow users to claim tokens manually.

Recommendation

Consider removing the automatic claims and force users to call the `claim` function manually.

Resolution

 RESOLVED

The team has stated: "Removed passive claiming, rewards can only be claimed when the `claim` function is explicitly called."

Issue #17**Unsafe casting within _updateUser function****Severity** INFORMATIONAL**Description**

_updateUser is called by the following functions:

- claim
- _beforeMint
- _beforeBurn
- _beforeBatchTransferFrom

The last three functions are invoked by the pair, hence the IDs parameter will be validated. For claim however, that is not the case. Theoretically, a user could provide an ID which will overflow on the uint24 cast and result in another also valid ID.

Fortunately, this will not have any negative side-effects since only the casted ID is used for further purposes.

Recommendation

Consider safe casting all ID parameters.

Safe casting should also be done for the uint256 casting of binStart and binEnd within _getRewardedRange and _getLiquidityDelta.

Resolution RESOLVED

Issue #18**Logic is not compatible with extraRewarder from Merchant Moe MasterChef****Severity** INFORMATIONAL**Description**

Merchant Moe's masterchef includes a potential extraRewarder:

```
function setExtraRewarder(uint256 pid, IMasterChefRewarder
extraRewarder) external override onlyOwner {
    _setExtraRewarder(pid, extraRewarder);
}
```

This contract however is not compatible with additional tokens, which would be stuck in the contract until swept by the admin.

Recommendation

Consider simply never adding an extraRewarder for these specific PIDs. If desired, this feature can be implemented in the future, however, this would require an additional audit round.

Resolution ACKNOWLEDGED

The team has stated:

"No extra rewarder will be added to the masterchef directly. The hook extra rewarder would be used instead."

Issue #19**Users can only claim as long as the rewarder is linked****Severity** INFORMATIONAL**Description**

The claim function includes a check which only allows users to claim as long as the rewarder is linked. If unlinked, users cannot claim their rewards anymore. This issue is only rated as informational because this is a security mechanism.

Recommendation

The team can simply acknowledge this issue.

Resolution ACKNOWLEDGED

The team has stated:

"In case of an event that force the removal of a rewarder, we would airdrop the unclaimed reward to user using our MerkleTree rewarder contract."

Issue #20**Sophisticated but unrealistic exploit allows for DoS in the pair under extreme situations****Severity** INFORMATIONAL**Description**

This issue is rated as informational because these conditions are very extreme and normally would not happen in practice. However, assuming the following:

- a) A token mintable by the token owner is used as reward
- b) The rewarder is linked to a masterchef which distributes this token instead of MOE (also works with MOE if mintable)
- c) The token owner swaps to bin 7501336 with a 128.128 price of 1
- d) The token owner mints a large amount of tokens to the contract, and due to the balance fetch within the `_updateRewards` call, this amount will be granted as reward for the coming state update

- e) The next state transition will increase the accumulator:

```
_bins[ids[i]].accRewardsPerShareX64 +=  
    pendingTotalRewards.mulDivRoundDown(weightX128,  
totalSupplyX64)
```

`totalSupplyX64` is 1 and `weightX128` is $1 \cdot (2^{128})$.

`pendingTotalRewards` is the large minted amount. Therefore, the addition reverts due to an overflow.

Recommendation

Consider only using trusted masterchefs and trusted tokens as reward tokens.

Resolution ACKNOWLEDGED

Issue #21	Typographical issues
Severity	INFORMATIONAL
Description	<u>L28</u> <pre>int256 internal constant MAX_NUBER_OF_BINS = 11;</pre> This should be named MAX_NUMBER_OF_BINS. <u>L277</u> <pre>/** * @dev Internal function to return whether the rewarder is linked * @return Whether the rewarder is linked */</pre> This comment is simply incorrect.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED



2.12 LB-Rewarder/LBHooksRewarder

LBHooksRewarder is similar to LBHooksExtraRewarder with the difference that rewards cannot be set arbitrarily. Instead, a sophisticated masterchef link is created which deposits in the underlying masterchef and receives rewards. Pending rewards are solely determined on the pending amount within the masterchef.

Optionally, it is possible to attach an extra rewarder for an additional reward stream, which will be invoked upon each state change.

2.12.1 Privileged Functions

- `setLBHooksExtraRewarder`



2.12.2 Issues & Recommendations

Issue #22	Broken MasterChef prevents state changes within LBPair
Severity	 INFORMATIONAL
Description	Whenever the state of the LBPair is changed (mint, burn, swap and transfer), <code>_updateReward</code> is invoked which harvests rewards from the masterchef. If the masterchef is broken and harvesting/updating pools revert, any pair state change will be DoSed until the hook is unplugged.
Recommendation	This issue is only rated as informational because hooks are designed exactly in such a way to unplug them if such issues occur.
Resolution	 ACKNOWLEDGED

Issue #23	Removal of ExtraRewarder will prevent users from claiming rewards
Severity	 INFORMATIONAL
Description	<code>setLBHooksExtraRewarder</code> allows the owner to remove an ExtraRewarder by setting <code>_extraHooksParameters</code> to zero. Since ExtraRewarders are never called directly by the user but solely via down-stream calls using the Hook library, any unclaimed rewards will be lost.
Recommendation	This is likely a design choice, therefore we do not recommend a change in the architecture. However, for consistency reasons we still need to note this issue.
Resolution	 ACKNOWLEDGED The team stated: "In case of an event that force the removal of a rewarder, we would airdrop the unclaimed reward to user using our MerkleTree rewarder contract."

Issue #24	Typographical issues
Severity	 INFORMATIONAL
Description	<u>L8</u> <pre>import {ILBPair} from "@lb-protocol/src/interfaces/ILBPair.sol";</pre> This import is unused. Consider removing it.
Recommendation	Consider fixing the typographical issue(s).
Resolution	 RESOLVED



2.13 LB-Rewarder/LBHooksExtraRewarder

LBHooksExtraRewarder contract is a trivial rewarder setup, which allows the contract owner to set an arbitrary reward rate. It includes a safety mechanism to not exceed the contract balance in the desired time-frame. The owner can change the rewardPerSecond, startTime and duration at any time.

It is solely meant to be invoked on top of the LBHooksRewarder.

2.13.1 Privileged Functions

- setRewarderParameters
- setRewardPerSecond



2.13.2 Issues & Recommendations

Issue #25	Unconsidered edge-case within <code>_getPendingTotalRewards</code> can result in DoS of reward update due to underflow
Severity	 HIGH SEVERITY
Description	<code>_getPendingTotalRewards</code> calculates the amount of rewards that have been allocated since the last update based on the time passed and the <code>_rewardsPerSecond</code> rate. A safety mechanism has been incorporated which ensures that after <code>endTimeStamp</code> there will be no more rewards being allocated: <pre>uint256 deltaTimestamp = block.timestamp < endTimeStamp ? block.timestamp - lastUpdateTimestamp : endTimeStamp - lastUpdateTimestamp;</pre> The problem with this safety mechanism is that it will underflow and revert whenever <code>lastUpdateTimestamp</code> is larger than <code>endTimeStamp</code>. Upon examination, this is a scenario which can happen in practice if the current <code>block.timestamp</code> exceeds <code>endTimeStamp</code>. In that scenario, <code>lastUpdateTimestamp</code> will be set to <code>block.timestamp</code>: <pre>if (block.timestamp > _lastUpdateTimestamp) _lastUpdateTimestamp = block.timestamp;</pre> This will then brick reward updates for any subsequent call, effectively DoS the pair completely until new reward parameters are set.
Recommendation	Consider returning early if <code>lastUpdateTimestamp</code> is larger than <code>endTimeStamp</code> .
Resolution	 RESOLVED

Issue #26	Typographical issues
Severity	 INFORMATIONAL
Description	<u>L4</u> <pre>import {IERC20} from "@openzeppelin/contracts/token/ERC20/ IERC20.sol";</pre> This import is unused and can be removed.
Recommendation	Consider fixing the typographical issue(s).
Resolution	 RESOLVED



2.14 Moe-Core/MasterChef

This audit is based on the following changes: <https://github.com/merchant-moe/moe-core/pull/9/files?diff=split&w=0#diff-50ed8da05f3802f765f3863771c60bd7f353cd6d86df056d5eb1da49dab72e7e>

The add function was changed to allow the HooksManager to add a pool. This change reflects the logic within the LBHooksRewarder contract.

No issues were found due to this change.

2.14.1 Issues & Recommendations

No issues found.



PALADIN
BLOCKCHAIN SECURITY