PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Trader Joe (MasterChef and Rewarder)

04 October 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

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

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

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

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

1 Overview

This report has been prepared for Trader Joe's MasterChef and Rewarder contracts on the Avalanche 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.

1.1 Summary

Project Name	Trader Joe
URL	https://traderjoexyz.com
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
SimpleRewarderPerSec	https://github.com/traderjoe-xyz/joe-core/blob/2a9f4887d0a7b591f879e16a4c29181bd8466f54/contracts/rewarders/SimpleRewarderPerSec.sol	
MasterChefJoeV3	https://github.com/traderjoe-xyz/joe-core/blob/2a9f4887d0a7b591f879e16a4c29181bd8466f54/contracts/MasterChefJoeV3.sol	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	2	2	-	-
Low	4	4	-	-
Informational	8	3	1	4
Total	**16**	**11**	**1**	**4**

Classification of Issues

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

1.3.1 SimpleRewarderPerSec

ID	Severity	Summary	Status
01	INFO	Contract uses raw division	PARTIAL
02	INFO	`lpSupply` is based on Masterchef contract balance	ACKNOWLEDGED
03	INFO	`setRewardRate` lacks upper limit safeguards	ACKNOWLEDGED
04	INFO	`emergencyWithdraw` can be used to take out all reward tokens prematurely	ACKNOWLEDGED

1.3.2 MasterChefJoeV3

ID	Severity	Summary	Status
05	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
06	HIGH	`rewardDebt` is miscalculated in `deposit` and `withdraw`	RESOLVED
07	MEDIUM	Functions are not secure from reentrancy	RESOLVED
08	MEDIUM	Insufficient Joe tokens may cause deposits and withdrawals to revert	RESOLVED
09	LOW	Deposits and withdrawals may fail if an invalid Rewarder contract is used	RESOLVED
10	LOW	Adding an EOA or non-token contract as a pool will break updatePool, massUpdatePools and updateEmissionRate	RESOLVED
11	LOW	Excess minting of dummy tokens may result in skewed reward emissions	RESOLVED
12	LOW	Duplicated pools may be added to the Masterchef	RESOLVED
13	INFO	Contract uses raw subtraction and division	RESOLVED
14	INFO	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
15	INFO	`add`, `set`, `deposit`, `withdraw`, `harvestFromMasterChef` and `emergencyWithdraw` functions can be made `external`	RESOLVED
16	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED

2 Findings

2.1 SimpleRewardPerSec

The SimpleRewarderPerSec contract allows for stakers in certain boosted-rewards pools in the MasterchefV3 to receive additional third-party partner tokens whenever a user deposits or withdraws from the Masterchef contract. The reward rate can be set to any amount, and so allow for the owner to set varying reward rates to incentivize stakers.

Reward tokens are transferred directly to the user via the onJoeReward function. It should be noted that sufficient reward tokens have to be transferred to this contract to allow for rewards to be given out to stakers, and can be topped up at any time in the future if the continuation of boosted rewards is desired.

2.1.1 Privileged Operations

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

- `setRewardRate`
- `emergencyWithdraw`

2.1.2 Issues & Recommendations

Issue #01	Contract uses raw division
Severity	INFORMATIONAL
Description	Lines with raw division operator: 117, 146, 156, 174, 177 Although the risks of a division by zero is almost non-existent, it is generally considered best practice to use SafeMath's `div` rather than raw division.
Recommendation	Consider using SafeMath's `div` instead.
Resolution	PARTIALLY RESOLVED Some functions still use raw division.

Issue #02	`lpSupply` is based on Masterchef contract balance
Severity	INFORMATIONAL
Location	Lines 112 and 169 `uint256 lpSupply = lpToken.balanceOf(address(MC_V2));`
Description	The Rewarder contract calculates `lpSupply` by querying the MasterchefV2 contract balance. This can cause dilution of rewards when people accidentally send tokens to the Masterchef. A more accurate method would be to have `lpSupply` being calculated from user deposits and withdrawals instead of token balance in the Masterchef contract.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` of the Masterchef that keeps track of the total deposits. Each `lpToken.balanceOf(address(MC_V2))` query can then be replaced with this `lpSupply` as well.
Resolution	ACKNOWLEDGED

Issue #03	**`setRewardRate` lacks upper limit safeguards**
Severity	INFORMATIONAL
Description	Currently, the `tokenPerSec` can be set to any amount by the owner of the contract, allowing for (if they wish) the token reward rate to be set to an unreasonably high amount.
Recommendation	Consider adding a reasonable upper limit using a `maxTokenPerSec` variable and safeguarding the `setRewardRate` function like so: `require(_tokenPerSec <= maxTokenPerSec, "Too high");`
Resolution	ACKNOWLEDGED

Issue #04	**`emergencyWithdraw` can be used to take out all reward tokens prematurely**
Severity	INFORMATIONAL
Description	Although there would clearly be a lack of incentives for the owner to do so, they nonetheless have the ability to remove reward tokens from the Rewarder contract by calling `emergencyWithdraw`. Note that this does not put user funds at risk, and merely affects the reward token to be paid out to stakers.
Recommendation	Simply setting this contract behind a 1 day Timelock should be sufficient in allowing users to see that the reward tokens are being withdrawn.
Resolution	ACKNOWLEDGED The client has stated that this will only ever be used if the rewarder should end reward distributions prematurely, and refund the reward tokens back to the partner protocol.

2.2 MasterChefJoeV3

A dummy token will be deployed, which will be deposited into this V3 Masterchef. This token will then be deposited into a specific pool in the V2 Masterchef, such that this contract would then be the recipient of all Joe staking rewards and can then distribute that to V3 stakers accordingly. This is done because the Joe token is owned by the V2 Masterchef, and token ownership cannot be transferred to the V3 Masterchef.

This contract would also allow for an external Rewarder contract to provide additional non-Joe tokens to users, essentially providing an APY boost to the farms. Finally, there are no deposit or withdrawal fees in the V3 Masterchef contract.

2.2.1 Privileged Operations

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

- `add`
- `set`

2.2.2 Issues & Recommendations

Issue #05	**Severely excessive rewards issue when a token with a transfer tax is added**
Severity	HIGH SEVERITY
Description	When tokens with a transfer tax are added to the pools, this will result in significant excessive rewards. Due to the way the Masterchef handles rewards, rewards can be heavily inflated when the balance of the Masterchef no longer matches that of user deposits. This happens for example with transfer tax tokens. This flaw of the Masterchef has recently been exploited on a significant number of projects, all of which their native tokens went to $0 afterwards because the exploit resulted in a large number of native tokens being minted and dumped. This issue was also present in SushiSwap (the original Masterchef). Since they were never meant to have any tokens but LP tokens, it was not a problem there but has become a problem to projects who have started forking it for usage with less standard tokens.

Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	
	Note that by using this method, you can also add the specific transfer tax logic for the native token.

Resolution	RESOLVED

Issue #06	**`rewardDebt` is miscalculated in `deposit` and `withdraw`**
Severity	HIGH SEVERITY

Description	Line 269
	`user.rewardDebt = user.rewardDebt.add(amount.mul(pool.accJoePerShare) / ACC_JOE_PRECISION);`
	
	The `rewardDebt` calculation in the `deposit` function is mis-specified, and goes against the normal `rewardDebt` calculation in other Masterchefs. In its current state, this calculation results in excessively high `rewardDebt`, which ultimately causes users to receive reduced harvest rewards in the future.
	
	Line 298
	`user.rewardDebt = user.rewardDebt.sub(amount.mul(pool.accJoePerShare) / ACC_JOE_PRECISION);`
	
	As before, the `rewardDebt` calculation in the `withdraw` function is mis-specified. In its current state, this calculation results in reducing `rewardDebt`, which causes users to receive more harvest rewards in the future.

Recommendation	Consider implementing the following fixes:
	
	For deposit
	`user.amount = user.amount.add(amount);`
	`user.rewardDebt = user.amount.mul(pool.accJoePerShare).div(ACC_JOE_PRECISION);`
	
	For withdraw
	`user.amount = user.amount.sub(amount);`
	`user.rewardDebt = user.amount.mul(pool.accJoePerShare).div(ACC_JOE_PRECISION);`
	
	This ensures that user.amount accurately accounts for deposits and withdrawals, and that `rewardDebt` calculation is correctly specified.

Resolution	RESOLVED

Issue #07	Functions are not secure from reentrancy
Severity	MEDIUM SEVERITY
Description	The `deposit`, `withdraw` and `emergencyWithdraw` functions are susceptible to reentrancy, especially if ERC777 tokens are added to the contract. This may result in the contract being inadvertently exploitable, as was the case with the AMP token in Cream Finance just recently.
Recommendation	Consider adding the `nonReentrant` modifier to the deposit, `withdraw`, `emergencyWithdraw` functions, and reordering line items in the `deposit` (ensuring that tokens are transferred into the Masterchef before `user.amount` is updated) and withdrawal-related functions to ensure best safety practices are adhered to per the Check Effects Interactions pattern.
Resolution	RESOLVED

Issue #08	**Insufficient Joe tokens may cause deposits and withdrawals to revert**
Severity	MEDIUM SEVERITY

Location	Lines 260-265, 290-295
	`if (user.amount > 0) {`
	`    // Harvest JOE`
	`    uint256 pending = user.amount.mul(pool.accJoePerShare).div(1e12).sub(user.rewardDebt);`
	`    JOE.safeTransfer(msg.sender, pending);`
	`    emit Harvest(msg.sender, pid, pending);`
	`}`

Description	Currently, there are only ever Joe tokens if the `harvestFromMasterChef` function is manually called. Should there be insufficient Joe tokens to pay out pending rewards, then the deposit and withdraw functions may revert (see code snippet above).

Recommendation	Consider calling the `harvestFromMasterChef` function at the start of `deposit` and `withdraw`, like so:
	`function deposit(uint256 pid, uint256 amount) public {`
	`    harvestFromMasterChef();`
	`    updatePool(pid);`
	
	`function withdraw(uint256 pid, uint256 amount) public {`
	`    harvestFromMasterChef();`
	`    updatePool(pid);`
	This will ensure that Joe tokens are harvested from the V2 Masterchef in order to pay out pending rewards.

Resolution	RESOLVED

Issue #09	Deposits and withdrawals may fail if an invalid Rewarder contract is used
Severity	LOW SEVERITY

Location	Lines 272-274, 302-304, 326-328
	`IRewarder _rewarder = pool.rewarder;`
	`if (address(_rewarder) != address(0)) {`
	`  _rewarder.onJoeReward(msg.sender, user.amount);`

Description	In the `deposit`, `withdraw` and `emergencyWithdraw` functions, external calls are made to the Rewarder contract. Should that be an invalid contract, then the external call and thus these functions (as well as pendingTokens) would fail.
Recommendation	Kindly ensure that only valid, non-malicious Rewarder contracts will be used, as there is the ability to swap out the Rewarder contract in the `set` function. Additionally, if for whatever reason an invalid Rewarder contract was used, the aforementioned `set` function can be used to correct this. This issue will be marked as Resolved once the client acknowledges this.
Resolution	RESOLVED The client will ensure that only valid, non-malicious, audited Rewarder contracts are used.

Issue #10	**Adding an EOA or non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`**
Severity	LOW SEVERITY
Description	`updatePool` will always call `balanceOf(address(this))` on the token of the pool, and will fail if the token is not an actual token contract address.
Recommendation	Consider simply adding a test line in the `add` function. If the token does not exist, this will make sure the `add` function fails. `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED

Issue #11	**Excess minting of dummy tokens may result in skewed reward emissions**
Severity	LOW SEVERITY
Description	The dummy token is to be deposited into a specific, pre-determined pool in the V2 Masterchef, which will funnel all emission rewards from that pool to the stakers in V3 Masterchef. When the init function is called, dummy tokens are transferred from `msg.sender` to the V3 Masterchef, and then on to the V2 Masterchef. If there are absolutely no other dummy tokens in circulation, then the V3 Masterchef would receive 100% of the dummy token pool's emissions in the V2 contract. However, if for whatever reason, an additional party also has dummy tokens, then they will be able to deposit that into the V2 Masterchef and thus receive staking rewards, diluting the yields available to V3. The `init` function may also be called by any user, though there is no harm in doing so other than reverting, unless they too have dummy tokens.
Recommendation	Consider ensuring that there are no additional dummy tokens minted, apart from what will be used to call the `init` function. The simplest solution to this would be to renounce ownership of the dummy token contract after the appropriate minting amount has been done. Additionally, the `init` function should only be called by the owner, and thus should have the `onlyOwner` modifier implemented in the function.
Resolution	RESOLVED The `init` function is now only callable by the Owner, and the client will also ensure that no excess dummy tokens are minted.

Issue #12	Duplicated pools may be added to the Masterchef
Severity	LOW SEVERITY
Description	The add function allows for duplicate pools to be added, which would lead to dilution of rewards for stakers.

Recommendation	The addition of a modifier that checks for duplicate pools could help prevent this incident from occurring.
	
	`mapping(IBEP20 => bool) public poolExistence;`
	`modifier nonDuplicated(IBEP20 _lpToken) {`
	`  require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated");`
	`  _;`
	`}`
	
	Alternatively, you could account for this by adding in an `lpSupply` variable under `poolInfo`. This has the benefit of accounting for accurately accounting for deposits in the Masterchef.

Resolution	RESOLVED

Issue #13	Contract uses raw subtraction and division
Severity	INFORMATIONAL
Description	Lines 227-230 `uint256 lpPercent = 1000 - MASTER_CHEF_V2.devPercent() - MASTER_CHEF_V2.treasuryPercent() - MASTER_CHEF_V2.investorPercent();` Although the risk of underflows occurring is very low, it is generally considered best practice to use SafeMath's sub rather than raw subtraction. Lines with raw division operator: 203, 204, 206, 231, 232, 243, 244, 269, 298 Although the risks of a division by zero is almost non-existent, it is generally considered best practice to use SafeMath's div rather than raw division.
Recommendation	Consider using SafeMath's sub and div instead.
Resolution	RESOLVED

Issue #14	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs, the total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. Each `lpToken.balanceOf(address(this))` query can then be replaced with this `lpSupply` as well.
Resolution	ACKNOWLEDGED

Issue #15	`add`, `set`, `deposit`, `withdraw`, `harvestFromMasterChef` and `emergencyWithdraw` functions can be made `external`
Severity	INFORMATIONAL
Description	The above functions can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider making these functions `external`.
Resolution	RESOLVED

Issue #16	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `accJoePerShare` is based on the `lpSupply` variable. `accJoePerShare = accJoePerShare.add(joeReward.mul(ACC_JOE_PRECISION) / lpSupply);` However, if this `lpSupply` becomes a severely large value, precision errors may occur due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals. Additionally, there are various instances in the contract where either `ACC_JOE_PRECISION` or `1e12` is used. For consistency purposes, either one of the other should be used exclusively within the contract.
Recommendation	Consider increasing the precision of `ACC_JOE_PRECISION` to `1e18`, and using that across the contract.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY