



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Singular Farm

31 October 2021



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 MasterSing	6
2 Findings	7
2.1 MasterSing	7
2.1.1 Issues & Recommendations	8

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.


1 Overview

This report has been prepared for Singular Farm 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	Singular Farm
URL	https://singular.farm/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
MasterSing	0xF2599B0c7cA1e3c050209f3619F09b6daE002857	 MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	3	0	3	-
● Medium	1	0	-	1
● Low	4	1	-	3
● Informational	7	2	-	5
Total	15	3	3	9

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 MasterSing

ID	Severity	Summary	Status
01	HIGH	initiate can be called again to potentially override the native token address and drain the Masterchef of any non-staked funds	PARTIAL
02	HIGH	Misconfiguration of key variables could lead to all withdrawals including emergency withdrawals reverting	PARTIAL
03	HIGH	Contract references a nonexistent address for the JoeRouter variable causing any singular buybacks to fail	PARTIAL
04	MEDIUM	earnedDebt is not reset in emergencyWithdraw	ACKNOWLEDGED
05	LOW	feeAddress is private	ACKNOWLEDGED
06	LOW	Usage of IBEP20 on Avalanche	RESOLVED
07	LOW	The pendingSing function will revert if totalAllocPoint is zero	ACKNOWLEDGED
08	LOW	If withdrawals ever break on Trader Joe, they will also break on Singular	ACKNOWLEDGED
09	INFO	WL_master, WL_earn and JoeRouter can be made constant	ACKNOWLEDGED
10	INFO	Contract does not support reflective tokens	RESOLVED
11	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
12	INFO	initiate, add, set, deposit, withdraw, emergencyWithdraw, dev, setFeeAddress and updateEmissionRate can be made external	ACKNOWLEDGED
13	INFO	Lack of events for add, set, dev, setFeeAddress and updateEmissionRate	ACKNOWLEDGED
14	INFO	msg.sender is unnecessarily cast to address(msg.sender)	ACKNOWLEDGED
15	INFO	Non-inclusive lesser than check within updateEmissionRate	RESOLVED

2 Findings



2.1 MasterSing

The MasterSing Masterchef is a fork of Goose Finance's Masterchef. The most notable feature of the MasterSing contract compared with traditional Masterchefs is that funds are not just idle within the contract. Instead, the deposited funds are staked within Trader Joe to receive another level of yield, making it a crossover between a Masterchef and a vault.

One of the important improvements that Singular has incorporated is to limit the deposit fees to 4%. In case Trader Joe ever enables dual rewards on pools, these dual rewards would be used to buyback and burn Singular tokens.



2.1.1 Issues & Recommendations

Issue #01	initiate can be called again to potentially override the native token address and drain the Masterchef of any non-staked funds
Severity	 HIGH SEVERITY
Location	<u>Lines 62-63</u> <pre>function initiate(SingToken _sing,address _devaddr,address _feeAddress,uint256 _singPerSec,uint256 _startTime) public onlyOwner{ require(_startTime>block.timestamp poolInfo.length==0,"start block passed");</pre>
Description	<p>The initiate function is currently improperly written to allow the owner to call it multiple times. This function is responsible for setting pretty much all configuration variables including most importantly the native reward token. As the contract also allows for pools which do not stake into Trader Joe, the Masterchef might contain funds of users. A malicious governance could then set the _sing native token to the address of a staked token, eg. USDC, and then the staked USDC could potentially be handed out as rewards (there are a few extra steps necessary for this exploit to circumvent the fact that minting would revert, but these steps are possible). The governance would likely have disabled all other pools except one in which they are the only staker to then receive all USDC within the Masterchef as rewards. This therefore allows the governance to drain the Masterchef of any token that is not staked in Trader Joe.</p>
Recommendation	<p>Consider fixing the guard on the initiate function to first of all use the startTime variable and second of all use && instead of .</p> <p>! Consider also adding parameter validation to also still require _startBlock to be in the future, the addresses to be non-zero and the singPerSec to be smaller or equal to one. Finally, if && is used, the for loop is no longer necessary.</p>
Resolution	 PARTIALLY RESOLVED <p>The client has indicated that they were aware of this shortcoming and that the relevant function is behind a timelock. However, since they cannot upgrade the contracts, it will only be fixed in subsequent versions of the contract.</p>

Issue #02

Misconfiguration of key variables could lead to all withdrawals including emergency withdrawals reverting

Severity

 HIGH SEVERITY

Description

Within the emergencyWithdraw function, special logic is added to still unstake the relevant funds from the Trader Joe Masterchef so they can be sent to the withdrawer. However, since this logic is quite complex, emergency withdrawals would fail if it is misconfigured. In general, this function should never fail.

1. isStrat and stratId should not be changeable on the pools.
2. earnFee should be capped to at most 100 within both set and add.
3. devAddr and feeAddress should not be settable to the zero address, since transfers to these addresses often revert (specifically the devAddr might break emergencyWithdraw).
4. Dual logic will revert if ever a bad rewarder is set by Trader Joe.

! The Sing token was excluded from the scope but if this contract contains complex logic on transfers it might furthermore break emergencyWithdraw due to the buybackSing function. We are assuming that it is a standard ERC-20 token and therefore never has unexpected behavior on transfer.

Recommendation

Consider fixing each of the individual points:

1. Remove isStrat and stratId from the set parameters
2. Cap earnFee to 100 within set and add using a requirement
3. Require devAddr and feeAddress to be non-zero in both the constructor and setters
4. Consider wrapping the rewarderBonusTokenInfo call in a try-catch clause. It should be noted that [such a clause is vulnerable to gas griefing](#). However, in this case we believe it would not be possible or profitable to execute a gas grief attack on this try-catch.

Resolution

 PARTIALLY RESOLVED

The client has indicated that they were aware of these shortcomings and they have put ownership behind a timelock (note: the devAddr and feeAddress are not). Since the contracts cannot be upgraded, they will only be fixed in subsequent versions of the contract.

Issue #03**Contract references a nonexistent address for the JoeRouter variable causing any singular buybacks to fail****Severity** HIGH SEVERITY**Location**Line 46

```
IUniswapV2Router02 public JoeRouter =  
IUniswapV2Router02(0xcF0feBd3f17CEf5b47b0cD257aCf6025c5BFf3b  
7);
```

Description

The contract contains a variable set to the address of the Trader Joe router. However, currently this address is set to the ApeSwap router on BSC causing any attempts to swap tokens on this router to revert. Furthermore, since the ApeSwap team can likely deploy any contract at this address because EVM-compatible addresses are deterministically based on the transaction nonce, this could potentially give ApeSwap extra privileges if they would deploy a malicious contract at this address.

! It should furthermore be noted that the Trader Joe V2 Masterchef had a vulnerability with the double rewards logic, as [disclosed per this article](#). It is unlikely that Trader Joe V2 will therefore ever contain double reward logic, which is heavily included within the Singular contract. TJ has been working on a V3 which properly manages double rewards, and our audit of this new contract can be found [here](#).

Recommendation

Consider using the correct address for the Trader Joe router, 0x60aE616a2155Ee3d9A68541Ba4544862310933d4. Consider also checking in with the Trader Joe team about their timeline for V3.

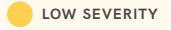

Resolution PARTIALLY RESOLVED

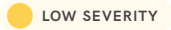
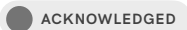
The client was already aware of this issue and will not be adding any double reward pools. As this issue is still present in the code, it has been marked as partially resolved. However, as long as no double reward pools are added (which is unlikely given the aforementioned article), users should not be affected in any way as the joeRouter is only used for these.


Issue #04	earnedDebt is not reset in emergencyWithdraw
Severity	● MEDIUM SEVERITY
Description	Within the emergencyWithdraw function, the user earnedDebt variable is not reset. This causes the UI function pendingEarned and a zero value withdrawal to revert.
Recommendation	Consider resetting earnedDebt similar to how the other variables are reset.
Resolution	● ACKNOWLEDGED

Issue #05	feeAddress is private
Severity	● LOW SEVERITY
Description	Important variables that third parties might want to inspect should be marked as public so that third parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variable as public.
Resolution	● ACKNOWLEDGED The client has indicated that users can still inspect this using the web3 getStorageAt function to discover private variables if they wish to do so.



Issue #06	Usage of IBEP20 on Avalanche
Severity	 LOW SEVERITY
Description	The contract uses the IBEP20 interface within Avalanche. However, within the Avalanche chain, we expect most coins to adhere to the slightly restricted ERC-20 interface.
Recommendation	Consider using the IERC20 interface instead. Consider also using SafeIERC20.
Resolution	 RESOLVED The client has indicated that BSC was their base chain and that they preferred to keep changes minimal while porting to the other chains. As these interfaces overlap almost completely and only the compatible functions are used, this does not affect users in any way.


Issue #07	The pendingSing function will revert if totalAllocPoint is zero
Severity	 LOW SEVERITY
Description	In the pendingSing function, at some point a division is made by the totalAllocPoint variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards since the lastRewardTime if the totalAllocPoint variable is greater than zero. This check can simply be added to the existing check that verifies the block.timestamp and pool.totalcap, like so: <pre>if (block.timestamp > pool.lastRewardTime && pool.totalcap != 0 && totalAllocPoint > 0) {</pre>
Resolution	 ACKNOWLEDGED The client has indicated they might add such a check in future version and that in the meantime they will never set totalAllocPoint to zero.

Issue #08**If withdrawals ever break on Trader Joe, they will also break on Singular****Severity** LOW SEVERITY**Description**

If a Masterchef messes up their rewards logic by either setting the `allocPoints` or minting rate exceptionally high and causing overflow reversions, all withdraw calls would fail. This is the reason every Masterchef including Trader Joe contains an `emergencyWithdraw` function which can be used to still withdraw in this situation.

Recommendation

To resolve this issue, a fundamental redesign of the system might be required similar to how panicable vaults work.

Resolution ACKNOWLEDGED


The client has indicated that this scenario is highly unlikely given the professionalism and scale of Trader Joe, however, they might add such an escape mechanism in the future.

Issue #09**WL_master, WL_earn and JoeRouter can be made constant****Severity** INFORMATIONAL**Description**

Variables that are never modified can be indicated as such with the `constant` keyword. This is considered best practice since it makes the code more accessible for third party reviewers and saves gas.

Recommendation

Consider making the above variables explicitly constant.

Resolution ACKNOWLEDGED

Issue #10**Contract does not support reflective tokens****Severity** INFORMATIONAL**Description**

The contract does not support tokens with a transfer tax. If these would be added, not everyone who deposits would be able to withdraw again as people would be withdrawing from each other's stake.

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);
```

This issue has been reduced to informational severity since the client clearly does not plan to support single-asset pools.

Resolution RESOLVED

The client has indicated that this was deliberately not done to preserve on gas costs. As the client basically only lists LP tokens, this issue has been marked as resolved.

Issue #11

Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions

Severity

INFORMATIONAL

Description

Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accSingPerShare` and `accEarnPerShare` is based upon the `pool.totalcap` variable.

```
pool.accSingPerShare =  
pool.accSingPerShare.add(singReward.mul(1e12).div(pool.total  
cap))
```

However, if this `pool.totalcap` becomes a severely large value, this will cause precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.

! Within the `updateReward` function, `_added.mul(pool.earnfee).div(100)` is also calculated twice.

Recommendation

Consider increasing precision to `1e18` across the entire contract. It should be noted that even a precision of `1e18` has been considered small when tokens like PolyDoge were added to Masterchefs of our client. In case the client thinks it is probable that such tokens will be added we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.

Also consider caching the redundant calculation in `updateReward` in a variable to reduce gas usage.

Resolution

ACKNOWLEDGED

The client has indicated that they also encountered this issue in production and will be careful to inspect the tokens they list in the future.

Issue #12

initiate, add, set, deposit, withdraw, emergencyWithdraw, dev, setFeeAddress and updateEmissionRate can be made external

Severity

 INFORMATIONAL

Description

Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. 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 marking the above variables as `external`.

Resolution

 ACKNOWLEDGED

The client has indicated that they will set these functions to `external` in subsequent contract versions.

Issue #13

Lack of events for add, set, dev, setFeeAddress and updateEmissionRate

Severity

 INFORMATIONAL

Description

Functions that affect the status of sensitive variables should emit events as notifications.

Recommendation

Add events for the above functions.

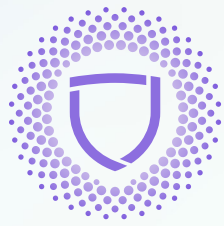
! Consider also renaming the `dev` function to `setDevAddress` to further improve the ease of interpretation of this function for third parties.

Resolution

 ACKNOWLEDGED

Issue #14	msg.sender is unnecessarily cast to address(msg.sender)
Severity	INFORMATIONAL
Description	The msg.sender is cast to address(msg.sender) throughout the contract when used with pool.IpToken.safeTransfer(). This is unnecessary.
Recommendation	Consider replacing all occurrences of address(msg.sender) with msg.sender.
Resolution	ACKNOWLEDGED

Issue #15	Non-inclusive lesser than check within updateEmissionRate
Severity	INFORMATIONAL
Location	<u>Line 340</u> require(_singPerSec<1 ether,"too large")
Description	The inequality check within the updateEmissionRate function is non-inclusive which in our experience tends to be undesirable by our clients as they might want to set such a variable to "up to one token per second" instead of 0.999... tokens per second.
Recommendation	Consider using <= instead.
Resolution	RESOLVED The client has indicated that the emission schedule was to start at 0.33 tokens per second and decrease this over time. This exclusive limit was only a safety guard and there are no plans to set the emission rate to exactly 1 token per second.



PALADIN
BLOCKCHAIN SECURITY