PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For WiseAvax

18 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 WiseAvax 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	WiseAvax
URL	https://wiseavax.finance/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`WiseAvax`	`0xcb341BE2ce94CA14686aB703212700D73Ac3341A`	MATCH
`MasterChef`	`0x226DDA03780Eb01860C46f7E1dfF9302b5bF1c30`	MATCH
`WiseAvaxTimelock`	`0x6c3D10782b6ef0b8121001baF2AF8E698E1dbC83`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	2	1	-	1
Informational	4	-	-	4
Total	**6**	**1**	**-**	**5**

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 WiseToken

ID	Severity	Summary	Status
01	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
02	INFO	`_totalSupply` is sufficient to keep track of the total token supply minted	ACKNOWLEDGED

1.3.2 MasterChef

ID	Severity	Summary	Status
03	LOW	Rewards are calculated based on block number instead of timestamp	ACKNOWLEDGED
04	INFO	`MAX_EMISSION_RATE` can be declared as a constant	ACKNOWLEDGED
05	INFO	Total token supply might not be minted due to try-catch pattern	ACKNOWLEDGED
06	INFO	Deposit uses raw subtraction instead of SafeMath for the before-after pattern	ACKNOWLEDGED

1.3.3 WiseAvaxTimelock

No issues found.

2 Findings

2.1 WiseToken

The WiseToken contract allows for Wise tokens to be minted when the `mint` function is called by the contract Owner, who at the time of deployment would be the deployer. Ownership is generally transferred to the Masterchef via the `transferOwnership` function for emission rewards to be minted and distributed to users staking in the Masterchef.

The `mint` function can be used to pre-mint tokens for various uses including injection of initial liquidity, token presale, airdrops, and others.

2.1.1 Token Overview

Address	0xcb341BE2ce94CA14686aB703212700D73Ac3341A
Token Supply	6,300
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None

2.1.2 Privileged Roles

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

- `mint`
- `renounceOwnership`
- `transferOwnership`

2.1.3 Issues & Recommendations

Issue #01	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	

Issue #02	**`_totalSupply` is sufficient to keep track of the total token supply minted**
Severity	INFORMATIONAL
Description	Both `MAXCAP` and `_totalSupply` are incremented by the amount of tokens minted, and decremented by the amount of tokens burnt. This makes `MAXCAP` redundant.
Recommendation	Consider removing `MAXCAP` and keeping track of the token supply using `_totalSupply`. If the emissions should stop when the `MAXCAP` is minted (eg. on burns, new emissions should not be possible again), consider removing the `MAXCAP` reduction on burn.
Resolution	

2.2 MasterChef

The MasterChef is a fork of Goose Finance's Masterchef. A notable feature of forking this Masterchef is the removal of the `migrator` function from the original Pancakeswap, which as of late has been used maliciously to steal user's tokens. Additionally, in comparison to Goose Finance, WiseAvax has limited the deposit fee to at most 4%. We commend WiseAvax on their decision to fork a relatively safer version of the Masterchef and trim down the governance privileges with regards to the deposit fees.

2.2.1 Privileged Roles

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

- `add`
- `set`
- `setDevAddress`
- `setFeeAddress`
- `updateEmissionRate`
- `updateStartBlock`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #03	**Rewards are calculated based on block number instead of timestamp**
Severity	LOW SEVERITY
Description	As rewards are calculated using `block.number` instead of `block.timestamp`, and block intervals are not always consistent on Fantom, it might be possible to accelerate the rewards beyond the expected emission rate by having blocks produced at a faster rate. This is a known issue for EVM-based chains such as Avalanche and Fantom. At the point of this review, it was observed that the average block time was 1 second.
Recommendation	Consider switching from calculation of rewards per block to rewards per second.
Resolution	

Issue #04	**`MAX_EMISSION_RATE` can be declared as a `constant`**
Severity	INFORMATIONAL
Description	As `MAX_EMISSION_RATE` is only declared once as a state variable and never changed, it can be declared as a constant for gas optimization.
Recommendation	Add the `constant` keyword when declaring `MAX_EMISSION_RATE` .
Resolution	

Issue #05	Total token supply might not be minted due to try-catch pattern
Severity	INFORMATIONAL

Description	As there is a MAXCAPSUPPLY for the Wise token, minting the reward and causing the maximum cap to exceed would result in a revert.
	
	`require(MAXCAP.add(amount) <= MAXCAPSUPPLY, "Max supply reached");`
	
	To prevent this, the following try and catch pattern is done in `updatePool`.
	
	`try wise.mint(devaddr, wiseReward.div(10)) { } catch (bytes memory reason) { wiseReward = 0; emit WiseMintError(reason); }`
	
	`try wise.mint(address(this), wiseReward) { } catch (bytes memory reason) { wiseReward = 0; emit WiseMintError(reason); }`
	
	In the case where `MAXCAP + amount` does exceed `MAXCAPSUPPLY`, the mint will not be done. This means that the token supply could be capped at an amount slightly lower than `MAXCAPSUPPLY`.

Recommendation	Consider minting the difference between `MAXCAPSUPPLY` and `MAXCAP`, if any.
Resolution	

Issue #06	Deposit uses raw subtraction instead of SafeMath for the before-after pattern
Severity	INFORMATIONAL

Location	Lines 1233-1235
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);`
	`_amount = pool.lpToken.balanceOf(address(this)) - balanceBefore;`

Description	The `deposit` function uses raw subtraction which could theoretically underflow since the contract is compiled on a Solidity version lower than 0.8.0. If a token which has special transfer logic that decreases the receivers' balance is added, the Masterchef can be drained of such token due to the underflow.
Recommendation	Consider using SafeMath instead: `_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
Resolution	

2.3 WiseAvaxTimelock

The Timelock contract is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	6 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	4 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.3.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY