PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Waterfall Finance

12 January 2022


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.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Waterfall Finance on the Fantom Opera 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	Waterfall Finance
URL	https://app.defiwaterfall.com/
Platform	Fantom Opera
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`Waterfall`	`0x6b2a7B82d3F7a6e1F5A5831aB40666Ec717645d5`	MATCH
`MasterChef`	`0x4Be7079064537867b40829119Be49Ee8CC76570e`	MATCH
`Timelock`	`0xeDDbcaDB0C9Ece20588226b087B4AE1Bf2861405`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	3	3	-	-
Low	3	2	-	1
Informational	4	-	-	4
Total	**10**	**5**	**-**	**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 Waterfall

ID	Severity	Summary	Status
01	MEDIUM	Sensitive state variable modifying functions can be changed instantaneously	RESOLVED
02	MEDIUM	Masterchef needs to be whitelisted in `_excludedFromAntiWhale` or large harvest amounts will revert	RESOLVED
03	LOW	Inconsistency between usage of `_msgSender()` and `msg.sender`	ACKNOWLEDGED
04	INFO	`lpEarner` can be replaced with `burnAddr`	ACKNOWLEDGED
05	INFO	Public functions can be made `external`	ACKNOWLEDGED

1.3.2 MasterChef

ID	Severity	Summary	Status
06	MEDIUM	Lack of non-zero address check in the constructor for `devAddr`	RESOLVED
07	LOW	Initial reward emission set in constructor can be higher than `MAX_EMISSION_RATE`	RESOLVED
08	LOW	Lack of non-zero address check in the constructor for `feeAddr`	RESOLVED
09	INFO	Unnecessary casting of `msg.sender` into `address`	ACKNOWLEDGED
10	INFO	State variables can be set as `immutable` for gas optimization	ACKNOWLEDGED

1.3.3 Timelock

No issues found.

2 Findings

2.1 Waterfall

The Waterfall token is an ERC-20 token which will be used as the main reward token for the Masterchef. It allows for Waterfall tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the Waterfall team. The ownership of the contract should be transferred to the Masterchef upon deployment of the contract. There is a maximum supply of 80,000 Waterfall tokens.

There is a maximum amount per transaction. If a transfer greater than this amount is made, and the sender or recipient is not in `_excludedFromAntiWhale`, the transaction will revert and not be successfully completed. The minimum capped for this is 0.5% of the supply.

For each non-whitelisted transfer, a maximum of 20% fees are enforced, and includes the burn fee and the liquidity fee. The fee percentages can be changed by the operator.

Whitelisted addresses are determined as the following:

- If recipient is the burn address
- If sender is operator
- If sender is one of the `noTaxSenderAddr`
- If recipient is one of the `noTaxRecipientAddr`

The liquidity fee is used in `swapAndLiquify` when the contract has at least more Waterfall tokens than `minAmountToLiquify`. Half of the `minAmountToLiquify` amount of Waterfall tokens is swapped to FTM, and 20% of the received FTM is sent to a hardcoded EOA fee address. The remaining FTM is paired with

`minAmountToLiquify` amount of Waterfall tokens and added as liquidity. The added liquidity pair tokens will be sent to the same address as the burn address.

As 20% of the FTM received is sent out to the fee address, all of the FTM received from the swap will be used, and there should not be any FTM dust in the token contract.

The burn fee is transferred to the burn address.

2.1.1 Token Overview

Address	0x6b2a7B82d3F7a6e1F5A5831aB40666Ec717645d5
Token Supply	80,000
Decimal Places	18
Transfer Max Size	Lower cap of 0.5 % of supply
Transfer Min Size	None
Transfer Fees	Up to 20%

2.1.2 Privileges

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

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

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

- `updateTransferTaxRate`
- `updateBurnFee`
- `updateMaxTransferAmountRate`
- `updateMinAmountToLiquify`
- `setExcludedFromAntiWhale`
- `updateSwapAndLiquifyEnabled`
- `updateSwapRouter`
- `setNoTaxSenderAddr`
- `setNoTaxRecipientAddr`
- `transferOperator`

2.1.3 Issues & Recommendations

Issue #01	**Sensitive state variable modifying functions can be changed instantaneously**
Severity	MEDIUM SEVERITY
Description	The following functions can be called to change state variables without any timelock: - `updateTransferTaxRate` - `updateBurnFee` - `updateMaxTransferAmountRate` - `updateMinAmountToLiquify` - `setExcludedFromAntiWhale` - `updateSwapAndLiquifyEnabled` - `updateSwapRouter` - `setNoTaxSenderAddr` - `setNoTaxRecipientAddr` Each of these functions can result in behavior that differs from expectations on how the token contract should behave.
Recommendation	The operator of the contract should be set to a reasonably delayed timelock to allow users to react to any upcoming changes.
Resolution	RESOLVED The timelock contract is now the operator. https://ftmscan.com/tx/0x59f2b2d535e59a1ed1a58f5170fae08d06781da3ec5033b155d3a69d213e801f

Issue #02	**Masterchef needs to be whitelisted in `_excludedFromAntiWhale` or large harvest amounts will revert**
Severity	MEDIUM SEVERITY
Description	If Masterchef is not whitelisted in `_excludedFromAntiWhale` and a harvest amount which exceeds `maxTransferAmount` is done, the transaction will revert. This would result in users being unable to make normal deposit or withdrawals from the Masterchef.
Recommendation	Ensure that the Masterchef contract address is whitelist in `_excludedFromAntiWhale`.
Resolution	RESOLVED The Masterchef has been excluded from anti-whale. https://ftmscan.com/tx/0x1bca7793b7e9461904d436eb510e540e709a58f70510d776c30e9dd6bfc19008

Issue #03	Inconsistency between usage of `_msgSender()` and `msg.sender`
Severity	LOW SEVERITY

Location	**Examples**
	Line 1313
	`modifier onlyOperator() {`
	`    require(_operator == msg.sender, "operator: caller is not the operator");`
	`    _;`
	`}`
	
	Line 1347
	`_operator = _msgSender();`

Description	There are instances in the code where `msg.sender` is used, and others where `_msgSender()` is used. This inconsistency could cause some issues as `_msgSender()` returns `msg.sender` for regular transactions, but for meta transactions, it can be used to return the end user instead of the relayer.
Recommendation	Consider using `_msgSender()` to replace all instances of `msg.sender`.
Resolution	ACKNOWLEDGED

Issue #04	`lpEarner` can be replaced with `burnAddr`
Severity	INFORMATIONAL
Description	As the `lpEarner` address and `burnAddr` are both the same address (0x000000000000000000000000000000000000dEaD), the `lpEarner` address can be removed and replaced with the `burnAddr` state variable.
Recommendation	Replace `lpEarner` with `burnAddr` and remove the `lpEarner` state variable.
Resolution	ACKNOWLEDGED

Issue #05	**Public functions can be made `external`**
Severity	INFORMATIONAL
Description	The following functions can be changed from public to external - `updateTransferTaxRate` - `updateBurnFee` - `updateMaxTransferAmountRate` - `updateMinAmountToLiquify` - `setExcludedFromAntiWhale` - `updateSwapAndLiquifyEnabled` - `updateSwapRouter` - `transferOperator` 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 (https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices)
Recommendation	Change the visibility of the above functions to `external`.
Resolution	ACKNOWLEDGED

2.2 MasterChef

The Waterfall Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking the latter is the removal of the `migrator` function from Sushiswap, which can potentially be used to steal user's tokens. Waterfall has limited the deposit fee to at most 4%.

Emissions have been modified to be per second instead of per block. The Masterchef has a check that ensures that only a maximum of 80,000 Waterfall tokens are minted. The maximum emission that can be set is 5 tokens per second. On top of the normal emissions, an additional 10% of emissions are minted to the dev address.

2.2.1 Privileges

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

- `add`
- `set`
- `updateEmissionRate`
- `updateStartTime`
- `transferOwnership`
- `renounceOwnership`

The following functions can be called by the DevAddr:

- `setDevAddr`

The following functions can be called by the FeeAddr:

- `setFeeAddr`

2.2.2 Issues & Recommendations

Issue #06	Lack of non-zero address check in the constructor for `devAddr`
Severity	MEDIUM SEVERITY
Description	Although the `devAddr` is checked to not be the zero address in `setDevAddress`, it is not done in the constructor. If the contract is deployed with the `devAddr` set as the zero address, all `updatePool` calls during will revert once emissions have started as the reward token does not allow minting to the zero address.
Recommendation	Consider adding the same non-zero address check for the `devAddr` in the constructor.
Resolution	RESOLVED The contract was deployed with a non-zero address as the `devAddress`.

Issue #07	Initial reward emission set in constructor can be higher than `MAX_EMISSION_RATE`
Severity	LOW SEVERITY
Description	Although the `updateEmissionRate` function ensures that the emission does not exceed `MAX_EMISSION_RATE`, it is not done in the constructor.
Recommendation	Consider adding the same check for the emission to be less than the maximum emission rate in the constructor.
Resolution	RESOLVED The contract was deployed with an emission below `MAX_EMISSION_RATE`.

Issue #08	Lack of non-zero address check in the constructor for `feeAddr`
Severity	LOW SEVERITY
Description	Although the `feeAddr` is checked to not be the zero address in `setFeeAddress`, it is not done in the constructor. If the contract is deployed with the `feeAddr` set as the zero address, deposits could revert for pools that have a deposit fee, if the deposit token does not allow the recipient address of a transfer to be the zero address.
Recommendation	Consider adding the same non-zero address check for the `feeAddr` in the constructor.
Resolution	RESOLVED The contract was deployed with a non-zero address as the `feeAddress`.

Issue #09	Unnecessary casting of `msg.sender` into `address`
Severity	INFORMATIONAL
Location	Line 1969 (example) `pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);`
Description	There are a number of instances where `msg.sender` is cast into an `address`, when it already is an `address` datatype.
Recommendation	Remove the casting of `msg.sender` as an `address`.
Resolution	ACKNOWLEDGED

Issue #10	State variables can be set as `immutable` for gas optimization
Severity	INFORMATIONAL
Description	The following state variable(s) can be set as `immutable` as they are set once in the constructor and never changed: - `waterfall`
Recommendation	Consider setting the above variable as `immutable`.
Resolution	ACKNOWLEDGED

2.3 Timelock

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. This contract should be the owner of the Masterchef contract to time delay making sensitive changes such as adding a new pool, or changing the allocation for an existing pool.

Parameter	Value	Description
Delay	TBC	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	2 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