PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Dogira (Staking)

13 November 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 Dogira's staking contracts on the Polygon 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	Dogira (staking)
URL	https://www.dogira.net/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`DogiraPool`	`0xFb3B75f84cBb583287B57A4Af2A4b7f6D91d96Eb`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	1	1	-	-
Low	2	2	-	-
Informational	9	9	-	-
Total	**15**	**15**	**-**	**-**

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 DogiraPool

ID	Severity	Summary	Status
01	HIGH	`totalStaked` and `user.amount` reduction amount does not account for `earlyWithdrawFees`	RESOLVED
02	HIGH	Harvests are lost while locked up if user deposits more	RESOLVED
03	HIGH	Lack of reentrancy protection for harvest related functions	RESOLVED
04	MEDIUM	Early withdrawal fee circumventable through `emergencyWithdraw`	RESOLVED
05	LOW	`rewardsWithdrawalRequested` is not set to `false` in `rewardsWithdrawal`	RESOLVED
06	LOW	`skimStakeTokenFees` can be used to drain rewards if stake and reward token is the same as the stake token	RESOLVED
07	INFO	`initialize` can be called by anyone	RESOLVED
08	INFO	`emergencyWithdraw` does not adhere to checks effects interactions pattern	RESOLVED
09	INFO	`emergencyWithdraw` emits 0 amount in event	RESOLVED
10	INFO	`safeTransfer` can be used in `sweepToken`	RESOLVED
11	INFO	`poolInfo.lpToken`, `poolInfo.allocPoints` and `totalAllocPoint` are unnecessary	RESOLVED
12	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	RESOLVED
13	INFO	`harvestLockupBlocks` should always be lesser than `earlyWithdrawalBlocks`	RESOLVED
14	INFO	`canWithdrawWithoutLockup` and `canWithdrawWithoutPenalty` should use `endBlock` as `noPenaltyBlock` if `noPenaltyBlock` is greater	RESOLVED
15	INFO	Lack of functionality to modify `feeAddress`	RESOLVED

2 Findings

2.1 DogiraPool

The DogiraPool is a fork of the Masterchef but with only a single staking pool. Dogira has limited the deposit fee to at most 5%. Deposit fees are charged on the function parameter `_amount` instead of the actual amount received by the contract.

There is also an early deposit period of a maximum of 300,000 blocks and if a withdrawal is done before that, an early withdrawal fee of up to 2% can be deducted from the withdrawing amount. Harvests can be subjected to a lockup of a maximum of 1,300,000 blocks.

The deposit fee, early withdrawal fee, early withdrawal blocks, and harvest lockup block state variables can only be changed before the start block of the pool. Users are advised to verify these values in the contract before depositing.

As a security feature, the owner is able to block smart contracts from depositing into the contract by setting the `blockSmartContracts` to `true` in the `initialize` function.

2.1.1 Privileged Roles

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

- `setRewardPerBlock`
- `skimStakeTokenFees`
- `setEndBlock`
- `setStartBlock`
- `setDepositFee`
- `setEarlyWithdrawalFee`
- `setEarlyWithdrawalBlocks`
- `setHarvestLockupBlocks`
- `enableDeposits`
- `requestRewardsWithdrawal`
- `rewardsWithdrawal`
- `sweepToken`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	**totalStaked and user.amount reduction amount does not account for earlyWithdrawFees**
Severity	HIGH SEVERITY

Location	Line 296~
	`if(_amount > 0) {`
	`  if (earlyWithdrawalFee > 0 && !`
	`canWithdrawWithoutPenalty(user.blockStaked)) {`
	`  uint256 withdrawalFee = _amount * earlyWithdrawalFee`
	`/ 10000;`
	`  poolInfo.lpToken.safeTransfer(address(feeAddress),`
	`withdrawalFee);`
	`  _amount = _amount - withdrawalFee;`
	`  emit FeeTaken(feeAddress, withdrawalFee);`
	`  }`
	`  user.amount = user.amount - _amount;`
	`  poolInfo.lpToken.safeTransfer(address(msg.sender),`
	`_amount);`
	`  totalStaked = totalStaked - _amount;`
	`}`

Description	If there is an `earlyWithdrawalFee` imposed on the `withdraw`, the fee is deducted from `_amount`, and sent to the `feeAddress`. The `user.amount` is correctly reduced by `_amount`, but `totalStaked` is also reduced by `_amount`, which does not contain the deducted fees. This causes `totalStaked` to be greater than the actual amount of token balance in the contract. Over time, this causes `totalStaked` to grow larger and larger than the actual contract balance, thus affecting the calculation of `accRewardTokenPerShare`. This is the same for `user.amount`, which should be reduced by the `_amount` before early withdraw fees, or the user would have more.

Recommendation

`totalStaked` should be reduced by the full amount, including the withdrawal fee, if any.

```
if(_amount > 0) {
    totalStaked -=  _amount;
    user.amount -= _amount;
    if (earlyWithdrawalFee > 0 && !
canWithdrawWithoutPenalty(user.blockStaked)) {
        uint256 withdrawalFee = _amount * earlyWithdrawalFee
/ 10000;

        poolInfo.lpToken.safeTransfer(address(feeAddress),
withdrawalFee);
        _amount = _amount - withdrawalFee;
        emit FeeTaken(feeAddress, withdrawalFee);
    }

    poolInfo.lpToken.safeTransfer(msg.sender, _amount);
}
```

Resolution

RESOLVED

The `totalStaked` and `user.amount` is now reduced by the correct `_amount`, before the subtract of withdrawal fees.

Issue #02	Harvests are lost while locked up if user deposits more
Severity	HIGH SEVERITY
Description	`rewardDebt` is updated regardless whether the harvest was successfully done. If the harvest is not done, it will result in the user losing the pending harvests. This can occur by subsequent deposit done to a pool.
Recommendation	If rewards are not harvested, they should be incremented in a variable (e.g. `lockedReward`) in `UserInfo`. When a successful harvest is done, the total sum should be the current `pending + lockedReward`. Also, `lockedReward` should be set to 0 in `emergencyWithdraw`.
Resolution	RESOLVED A new field, `UserInfo.lockedDebt`, has been added. In `deposit`, if additional deposits are done and the user is still not eligible for harvest, the pending amount would be added to `lockedDebt`. Otherwise, if the user is eligible for harvest, the sum of the `pending` and `lockedDebt` will be transferred to the user. The user's `lockedDebt` will then be set to zero. In `withdraw`, which can only be done when the user is eligible for harvest, the sum of the `pending` and `lockedDebt` will be transferred to the user. Similar to deposit's harvest, the user's `lockedDebt` will be set to zero. If the user does an `emergencyWithdraw`, `lockedDebt` will be set to zero.

Issue #03	Lack of reentrancy protection for harvest related functions
Severity	HIGH SEVERITY
Description	If the rewards token used has some form of callback functionality on transfer, it would be possible to conduct a reentrancy attack to drain the rewards as the `rewardDebt` is only updated after the transfer of the reward token.
Recommendation	Add `nonReentrant` modifiers to `deposit`, `withdraw` and `emergencyWithdraw`.
Resolution	RESOLVED The `nonReentrant` modifier has been added for `deposit`, `withdraw` and `emergencyWithdraw`.

Issue #04	Early withdrawal fee circumventable through `emergencyWithdraw`
Severity	MEDIUM SEVERITY
Description	If the harvest lock block has passed, but not the early withdraw fee block, it is possible for a user to harvest followed by `emergencyWithdraw` to circumvent the early withdraw fee.
Recommendation	Add the same logic check for early withdraw fee in `emergencyWithdraw` too.
Resolution	RESOLVED The same logic check for early `withdraw` fee has been added to `emergencyWithdraw`.

Issue #05	**`rewardsWithdrawalRequested` is not set to `false` in `rewardsWithdrawal`**
Severity	LOW SEVERITY
Description	`rewardsWithdrawal` requires `requestRewardsWithdrawal` to have been called before to set `rewardsWithdrawalRequested` to `true`, but `rewardsWithdrawalRequested` is not reset to `false` after `rewardsWithdrawal` is successfully executed.
Recommendation	Set `rewardsWithdrawalRequested` after the `require` statement.
Resolution	RESOLVED `rewardsWithdrawalRequested` is now set to false in `rewardsWithdrawal`.

Issue #06	**`skimStakeTokenFees` can be used to drain rewards if stake and reward token is the same as the stake token**
Severity	LOW SEVERITY
Description	`skimStakeTokenFees` uses `getStakeTokenFeeBalance` to get the amount to skim, but if the stake and reward token are both the same, `stakeTokenFeeBalance` would include the reward balance.
Recommendation	Consider not allowing skimming if the stake and reward token are the same.
Resolution	RESOLVED The following check has been added to prevent calling `skimStakeTokenFees` if the reward and stake token are the same. `require(address(REWARD_TOKEN) != address(STAKE_TOKEN), "Cannot skim same-token pairs!");`

Issue #07	`initialize` can be called by anyone
Severity	INFORMATIONAL
Description	Instead of using a constructor to initialize the state variables, the contract uses an `initialize` function instead. As it does not have the `onlyOwner` modifier unlike other privileged functions in the same contract, it is possible for anyone to call the `initialize` function after the contract has been deployed.
Recommendation	Add the `onlyOwner` modifier to ensure that only the owner can call initialize.
Resolution	RESOLVED `initialize` is now `onlyOwner`.

Issue #08	`emergencyWithdraw` does not adhere to checks effects interactions pattern
Severity	INFORMATIONAL
Description	The `emergencyWithdraw` function does the external call to the `lpToken` contract first before making the changes to the state variables.

Recommendation	Consider following the checks effects interactions pattern
	`uint256 amount = user.amount;`
	`totalStaked = totalStaked - amount;`
	`user.amount = 0;`
	`user.rewardDebt = 0;`
	`poolInfo.lpToken.safeTransfer(msg.sender, amount);`
	`emit EmergencyWithdraw(msg.sender, amount);`

Resolution	RESOLVED The code has been reordered to cache the amount, make the state changes, and finally do the external `safeTransfer` call.

Issue #09	`emergencyWithdraw` emits 0 amount in event
Severity	INFORMATIONAL
Description	As the `EmergencyWithdraw` uses `user.amount` after it is set to 0, the amount in the event will always be 0 instead of the actual amount withdrawn.

Recommendation	Consider storing `user.amount` in a local variable and use it for the event.
	`uint256 amount = user.amount;`
	`[...]`
	`emit EmergencyWithdraw(msg.sender, amount);`

Resolution	RESOLVED The event now uses the cached `user.amount` value in a local variable to emit the correct value.

Issue #10	**`safeTransfer` can be used in `sweepToken`**
Severity	INFORMATIONAL
Description	As sweepToken uses the IERC20 interface which is using SafeERC20, `safeTransfer` can be used in case the token to be swept is not ERC20 compliant.
Recommendation	Modify `token.transfer` to `token.safeTransfer`.
Resolution	RESOLVED `safeTransfer` is now used in sweepToken.

Issue #11	**`poolInfo.lpToken`, `poolInfo.allocPoints` and `totalAllocPoint` are unnecessary**
Severity	INFORMATIONAL
Description	`lpToken` is the same as `STAKE_TOKEN`, and would be an unnecessary duplicate. `allocPoint` is the same as `totalAllocPoint`, both being `1000`, so the `allocPoint/totalAllocPoint` would always result in 1. The removal of these fields would also make `poolInfo.allocPoint / totalAllocPoint` in the calculation of `tokenReward` in `pendingReward` and `updatePool` redundant.
Recommendation	Remove `lpToken` and `allocPoint` from the `poolInfo` structure, and the `totalAllocPoint` state variable. The unnecessary arithmetic in `tokenReward` calculation can also be done as the following: `uint256 tokenReward = multiplier * rewardPerBlock;`
Resolution	RESOLVED The unnecessary variables have been removed.

Issue #12	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	`msg.sender` is cast to `address(msg.sender)` in some instances of the contract. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	RESOLVED The occurrences have been replaced.

Issue #13	`harvestLockupBlocks` should always be lesser than `earlyWithdrawalBlocks`
Severity	INFORMATIONAL
Description	If the `harvestLockupBlocks` is greater than `earlyWithdrawalBlocks`, it would defeat the purpose of `earlyWithdrawalBlocks`. All `withdraw` function calls would have required to pass the `harvestLockupBlock` of the user withdrawing, and a lesser `earlyWithdrawalBlocks` would mean that early withdrawal fees would never be deducted.
Recommendation	Consider adding a check in `initialize` and setter functions to ensure that `harvestLockupBlocks < earlyWithdrawalBlocks`.

Resolution	RESOLVED
	The following check has been added to ensure that `harvestLockupBlocks` is lesser than `earlyWithdrawalBlocks` if the `earlyWithdrawFee` is not zero:
	`if (earlyWithdrawalFee > 0) {`
	`require(_harvestLockupBlocks < earlyWithdrawalBlocks, 'harvestLockupBlocks must be less than earlyWithdrawalBlocks!');`
	`}`

Issue #14	**canWithdrawWithoutLockup and canWithdrawWithoutPenalty should use endBlock as noPenaltyBlock if noPenaltyBlock is greater**
Severity	INFORMATIONAL
Description	If a user deposits towards the nearing of the `endBlock`, harvesting or withdrawing would not be possible even after the `endBlock` has been passed, as long as the `noPenaltyBlock` is greater than `endBlock`.

Recommendation	Consider using `endBlock` as `noPenaltyBlock` if `noPenaltyBlock > endBlock`.
	`uint256 noPenaltyBlock = harvestLockupBlocks + _stakedInBlock;`
	`if (noPenaltyBlock > endBlock) {`
	`  noPenaltyBlock = endBlock;`
	`}`

Resolution	RESOLVED `noPenaltyBlock` will be set as the `endBlock` if it is greater than the `endBlock`.

Issue #15	**Lack of functionality to modify `feeAddress`**
Severity	INFORMATIONAL
Description	The `feeAddress` is set once in the `initialize` function and cannot be changed after that. If the `feeAddress` gets compromised (e.g. private key leak), the `feeAddress` will continue to receive fees and cannot be modified to an address controlled by the team.
Recommendation	Consider adding a function for the owner to modify the `feeAddress`. There should be a check to ensure that the `feeAddress` is not a zero address.
Resolution	RESOLVED An `updateFeeAddress` function has been added for the owner. The `feeAddress` has a check to not allow the zero-address from being set.


PALADIN
BLOCKCHAIN SECURITY