PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Daikiri Finance

28 July 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 Daikiri Finance. 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. This audit only covers one contract: MixologistMiner.sol

2.1 Summary

Project Name	Daikiri Finance
URL	https://daikiri.finance/
Platform	Harmony
Language	Solidity

2.2 Contract Assessed

The client has implemented our recommendations and provided the final contracts in a zip archive. Users should note that there are other contracts in the zip archive, but we have audited only MixologistMiner.

We have uploaded the zip archive containing the updated MixologistMiner.sol and will verify that the contents are a match once they have deployed them on the blockchain.

Name	Contract / File	Live Code Match
`MixologistMiner`	`MixologistMiner.sol`	PENDING
`Final Zip Archive`	`https://paladinsec.co/assets/audits/source/MixologistMiner.zip`	

2.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	1	1	-	-
Low	6	4	2	-
Informational	14	7	-	7
Total	**24**	**15**	**2**	**7**

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.

2.3.1 Farm Section

ID	Severity	Summary	Status
01	HIGH	`updateStartBlock` does not update `PoolInfo.lastRewardBlock`	RESOLVED
02	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
03	MEDIUM	Tokens must be present in the Masterchef for the contract to function	RESOLVED
04	LOW	Adding an EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`	RESOLVED
05	LOW	`updateEmissionRate` has no maximum safeguard	PARTIALLY RESOLVED
06	LOW	The `pendingRewardToken` function will revert if `totalAllocPoint` is zero	RESOLVED
07	LOW	The referral contract could be changed by the owner to potentially steal referral rewards	RESOLVED
08	INFORMATIONAL	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
09	INFORMATIONAL	`deposit`, `withdraw`, `emergencyWithdraw`, mint, `checkMintSolution`, and `changeMiningReward` can be made `external`	RESOLVED
10	INFORMATIONAL	Lack of events for `add`, `set`, and `updateStartBlock`	RESOLVED
11	INFORMATIONAL	`setDaoAddress` doesn't actually do anything	RESOLVED
12	INFORMATIONAL	`add` function could allow for accidentally adding `stakingToken` as `rewardToken`	ACKNOWLEDGED
13	INFORMATIONAL	`PRECISION_FACTOR` and `rewardToken` can be made immutable	ACKNOWLEDGED
14	INFORMATIONAL	`MAXIMUM_HARVEST_INTERVAL` comment says it is 14 days while in reality it is just 24 hours	RESOLVED

2.3.2 Miner Section

ID	Severity	Summary	Status
15	HIGH	If a `challengeNumber` ever occurs twice, the system will irreversibly break	RESOLVED
16	LOW	Replay possible if there are multiple clones of this project	RESOLVED
17	LOW	Uncapped variable `miningReward`	PARTIALLY RESOLVED
18	INFORMATIONAL	Target blocks per epoch might not be clear as it is hard coded	ACKNOWLEDGED
19	INFORMATIONAL	Unused return value in `changeMiningReward`	RESOLVED
20	INFORMATIONAL	Mined event emits the new `epochCount` and new challenge number	ACKNOWLEDGED
21	INFORMATIONAL	Division before multiplication	ACKNOWLEDGED
22	INFORMATIONAL	Comment in the difficulty adjustment logic says the difficulty will be harder while it is actually easier in this section	RESOLVED
23	INFORMATIONAL	`_BLOCKS_PER_READJUSTMENT`, `_MINIMUM_TARGET` and `_MAXIMUM_TARGET` can be made constant	RESOLVED
24	INFORMATIONAL	Unused variable `challenge_digest` in helper function `getMintTest`	ACKNOWLEDGED

2 Findings

2.1 MixologistMiner

The MixologistMiner contract contains both a Masterchef for normal yield farming staking, as well as a Proof of Work (POW) concept that was taken from 0xBitcoin that allows users to participate in mining for rewards. There are no deposit fees when staking in the contract. All reward tokens that will be given out for the Masterchef and POW mining are to be deposited into the MixologistMiner contract for distribution as there is no token minting ability. The benefit of this is that the contract allows for any token, not just the native tokens, to be given out as rewards.

The Proof Of Work concept allows users to direct computer power and mining software to solve ever-changing difficulty 'Challenges'. The difficulty may be higher or lower depending on past mining efforts, and is readjusted every 1024 reward blocks. If a solution matches the Challenge, then reward tokens are sent directly to the miner.

2.1.1 Farm Section: Issues & Recommendations

Issue #01	`updateStartBlock` does not update `PoolInfo.lastRewardBlock`
Severity	HIGH SEVERITY
Description	When users deposit in a pool, they will start earning rewards as soon as the current block is larger than the pools' `lastRewardBlock`. When a pool is added, this `lastRewardBlock` is set to either the `currentBlock` or the `startBlock` (whichever is greater). If the `startBlock` is moved, the `lastRewardBlock` absolutely has to be moved as well. There have been multiple cases of projects that have failed to start as their `lastRewardBlock` was still weeks in the future, even though they called `updateStartBlock`. This issue can cause significant reputational damage.
Recommendation(s)	Consider adding a loop to update all `lastRewardBlocks` in the `updateStartBlock` function. Note that this loop may run out of gas if a very significant amount of pools are added.

```
function updateStartBlock(uint256 _startBlock) external onlyOwner {
  require(startBlock > block.number, "Farm already started");
  uint256 length = poolInfo.length;

  for (uint256 pid = 0; pid < length; ++pid) {
    PoolInfo storage pool = poolInfo[pid];
    pool.lastRewardBlock = _startBlock;
  }

  startBlock = _startBlock;
}
```

Resolution	RESOLVED

Issue #02	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, there will be significant excessive rewards due to the way the Masterchef handles the reward mechanism: 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 has recently been exploited on a significant number of projects, all of which their native tokens went to $0 afterwards. 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(s)	Consider using the current standard of handling deposits, which is based on how uniswap handles transfer fees:

```
uint256 balanceBefore = pool.stakingToken.balanceOf(address(this));
pool.stakingToken.transferFrom(msg.sender, address(this), _amount);
_amount = pool.stakingToken.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 #03	Tokens must be present in the Masterchef for the contract to function
Severity	MEDIUM
Description	Contrary to most Masterchefs, this Masterchef is not the owner of the token and thus does not mint the token itself. Instead, it will simply try to transfer tokens that are in its balance. In the case that there are insufficient tokens, native deposits will be depleted first (resulting in the transfer tax issue) and then no more rewards will be given.
Recommendation(s)	First, ensure that the Masterchef will have enough tokens. Second, keep track of the total deposits in the native pool and ensure that the native balance cannot become lower than this value when rewards are given out.
Resolution	RESOLVED Reward tokens will be minted by the MixologistMiner contract.

Issue #04	Adding an EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`
Severity	LOW SEVERITY
Description	updatePool will always call `balanceOf(address(this))` on the token of this pool.
Recommendation(s)	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_stakingToken.balanceOf(address(this));`
Resolution	RESOLVED

Issue #05	`updateEmissionRate` has no maximum safeguard
Severity	LOW SEVERITY
Description	Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent.
Recommendation(s)	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_rewardTokenPerBlock <= MAX_EMISSION_RATE,"Too high emission");`
Resolution	PARTIALLY RESOLVED This has been partially resolved as we are unable to verify the value for `MAX_EMISSION_RATE` until the contracts have been deployed.

Issue #06	The `pendingRewardToken` function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	In the `pendingRewardToken` 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(s)	Consider only calculating the accumulated rewards since the `lastRewardBlock` of the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the block number and `stakedTokenSupply`.
Resolution	RESOLVED

Issue #07	The referral contract could be changed by the owner to potentially steal referral rewards
Severity	LOW SEVERITY
Description	The owner could set the referral contract to a new contract that always returns their address as the referral of a user. This would allow them to take up to a maximum of 5% of the minting rewards since that is the referral reward limit.
Recommendation(s)	Consider removing the upgradeability of the referral contract and simply setting it in the constructor.
Resolution	RESOLVED

Issue #08	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. This dilution is amplified even a bit more because the native token in question is a reflection token.
Recommendation(s)	Consider adding an `stakingTokenSupply` variable to the PoolInfo that keeps track of the total deposits.
Resolution	ACKNOWLEDGED

Issue #09	`deposit`, `withdraw`, `emergencyWithdraw`, `mint`, `checkMintSolution`, and `changeMiningReward` can be made `external`
Severity	INFORMATIONAL
Description	`deposit`, `withdraw`, `emergencyWithdraw`, `mint`, `checkMintSolution`, and `changeMiningReward` 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 (https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices).
Recommendation(s)	Consider making these functions external.
Resolution	RESOLVED

Issue #10	Lack of events for `add`, `set`, and `updateStartBlock`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation(s)	Add events for `add`, `set`, and `updateStartBlock`.
Resolution	RESOLVED

Issue #11	`setDaoAddress` doesn't actually do anything
Severity	INFORMATIONAL
Description	`DaoAddress` doesn't have any privileges in the contract and thus serves no purpose.
Recommendation(s)	Consider removing this if there are no plans to use it.
Resolution	RESOLVED

Issue #12	`add` function could allow for accidentally adding `stakingToken` as `rewardToken`
Severity	INFORMATIONAL
Description	Setting `_stakingToken` to be the same as `rewardToken` may lead to undesirable results.
Recommendation(s)	Consider adding the following to the `add` function: `require(_stakingToken != _rewardToken);`
Resolution	ACKNOWLEDGED

Issue #13	`PRECISION_FACTOR` and `rewardToken` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are set in the constructor but remain unchanged afterwards can be marked as immutable. This is good practice as it signals this behavior to third-party reviewers, making their lives easier.
Recommendation(s)	Consider marking the mentioned variables as immutable.
Resolution	ACKNOWLEDGED

Issue #14	`MAXIMUM_HARVEST_INTERVAL` comment says it is 14 days while in reality it is just 24 hours
Severity	INFORMATIONAL
Description	A comment above the `MAXIMUM_HARVEST_INTERVAL` definition indicates that it is set to 14 days while in reality this limit is set to 24 hours. We believe investors will be more comfortable with the actual limit so we do not see a reason to advertise 14 days.
Recommendation(s)	Consider removing the comment in question.
Resolution	RESOLVED The `MAXIMUM_HARVEST_INTERVAL` is now 6 hours.

2.1.2 Miner Section: Issues & Recommendations

Issue #15	If a `challengeNumber` ever occurs twice, the system will irreversibly break
Severity	HIGH SEVERITY
Description	If a `challengeNumber` has already received a solution, the system does not allow this `challengeNumber` to be mined again. Instead, the `mint` transaction will revert. Since the `challengeNumber` can only be updated through mining, the system will not be able to ever mine rewards after this. This issue is problematic since nothing is stopping someone to mint a reward twice in the same block. The `challengeNumber` the second time would be the same as the first time.
Recommendation(s)	There are two issues to solve here: 1. Ensure that the system can never enter the breaking state: If the solution already exists, instead of reverting, simply do not give a reward and generate a new challenge. 2. If you mine multiple times in the same block, make sure each time a different `challengeNumber` is generated by including the previous `challengeNumber` in the a hash:

```
challengeNumber = keccak256(abi.encodePacked(challengeNumber, address(this),
blockhash(block.number - 1));
```

Resolution	RESOLVED

Issue #16	Replay possible if there are multiple clones of this project
Severity	LOW SEVERITY
Description	If there are multiple copies of the MixologistMiner running, it might be possible to split mine. This is because the challengeNumber will be the same on all copies.
Recommendation(s)	Consider hashing in the contract address in the challengeNumber generation:

```
challengeNumber = keccak256(abi.encodePacked(challengeNumber, address(this),
blockhash(block.number - 1));
```

Resolution	RESOLVED

Issue #17	Uncapped variable `miningReward`
Severity	LOW SEVERITY
Description	The `miningReward` can be set to an excessive value by the owner, draining the Masterchef on the next mine.
Recommendation(s)	Consider adding a reasonable maximum to the `changeMiningReward` function.
Resolution	PARTIALLY RESOLVED This issue has been partially resolved as we are unable to verify the value for `MAX_MINING_REWARD` until the contracts have been deployed.

Issue #18	Target blocks per epoch might not be clear as it is hard coded
Severity	INFORMATIONAL
Description	Currently the target is to mine a reward every 60 blocks, however, this is set as a hardcoded variable in the code. We decided to point this out explicitly in the report to ensure that the team is aware of this.
Recommendation(s)	Consider whether 60 blocks per reward is a desirable average and consider moving this parameter as a constant variable to the top of the contract.
Resolution	ACKNOWLEDGED

Issue #19	Unused return value in `changeMiningReward`
Severity	INFORMATIONAL
Description	`changeMiningReward` has a `return` value but we are unsure why it is there as it is not used.
Recommendation(s)	Consider removing this value.
Resolution	RESOLVED

Issue #20	Mined event emits the new `epochCount` and new challenge number
Severity	INFORMATIONAL
Description	The Mined event actually emits the data for the next epoch; this might not be desirable behavior.
Recommendation(s)	Consider whether the Mined event should emit the data for the current event, the next event, or both.
Resolution	ACKNOWLEDGED

Issue #21	Division before multiplication
Severity	INFORMATIONAL
Description	In the reward adjustment logic (`_reAdjustDifficulty`), division is sometimes done before multiplication. Because of rounding errors during division, the end result will be less precise. It should be noted that this will not be a severe issue as the `miningTarget` is a large number anyways.
Recommendation(s)	Consider doing multiplication before division to increase the precision.
Resolution	ACKNOWLEDGED

Issue #22	**Comment in the difficulty adjustment logic says the difficulty will be harder while it is actually easier in this section**
Severity	INFORMATIONAL
Description	In the section that makes the difficulty easier, the comment `// Make it harder` is still present. This comment is presumably copied from the previous section but of course it should be easier in this section.
Recommendation(s)	Consider changing the comment to indicate that this is the section that makes the difficulty less hard.
Resolution	RESOLVED

Issue #23	**`_BLOCKS_PER_READJUSTMENT`, `_MINIMUM_TARGET` and `_MAXIMUM_TARGET` can be made constant**
Severity	INFORMATIONAL
Description	The constant miner variables can be explicitly declared as constant. This is best practice as it simplifies the reviewing process for third-party reviewers and static-analysis tools could also take this into consideration.
Recommendation(s)	Consider making these variables constant.
Resolution	RESOLVED

Issue #24	Unused variable `challenge_digest` in helper function `getMintTest`
Severity	INFORMATIONAL
Description	The `challenge_digest` variable is unused in the `getMintDigest` function.
Recommendation(s)	Consider removing this variable.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY