PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For PolyKiwi Finance

21 August 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 PolyKiwi 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.

1.1 Summary

Project Name	PolyKiwi
URL	https://polykiwi.finance/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

The contracts were provided to Paladin in a GitHub repository. We will update this report once we have verified that the deployed contracts match the ones we have audited.

Name	Contract	Live Code Match
`MasterChef`	`https://github.com/polykiwi-finance/polykiwi-farms/blob/master/contracts/MasterChef.sol`	
`Timelock`	`https://github.com/polykiwi-finance/polykiwi-farms/blob/master/contracts/Timelock.sol`	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	2	2	-	-
Low	4	4	-	-
Informational	8	7	1	-
Total	**17**	**16**	**1**	**0**

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 MasterChef

ID	Severity	Summary	Status
01	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
02	HIGH	`emergencyWithdraw` transfers no funds	RESOLVED
03	HIGH	User balances does not properly record deposits if there are no deposit fees	RESOLVED
04	MEDIUM	`payReferral` function can fail if there are insufficient tokens	RESOLVED
05	MEDIUM	Duplicated pools may be added to the Masterchef	RESOLVED
06	LOW	Setting `feeAddress` to the zero address will break most functionality	RESOLVED
07	LOW	The `pendingKiwi` function will revert if `totalAllocPoint` is zero	RESOLVED
08	LOW	Adding an EOA or a non-token contract as a pool will break `updatePool` and `massUpdatePools`	RESOLVED
09	LOW	`updateEmissionRate` has no maximum safeguard	RESOLVED
10	INFORMATIONAL	Native token and associated comments still reference CAKE (from PancakeSwap)	PARTIALLY RESOLVED
11	INFORMATIONAL	`BONUS_MULTIPLIER` looks to be redundant	RESOLVED
12	INFORMATIONAL	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
13	INFORMATIONAL	Various unused comments remain in the contract	RESOLVED
14	INFORMATIONAL	`kiwi` and `startBlock` can be made immutable	RESOLVED
15	INFORMATIONAL	Pools use the contract balance to figure out the total deposits	RESOLVED
16	INFORMATIONAL	`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `setFeeAddress`, updateEmissionRate, `toggleReferrals` can be made `external`	RESOLVED
17	INFORMATIONAL	Lack of events for `add`, `set`, `setFeeAddress`, `updateEmissionRate`, `toggleReferrals`	RESOLVED

1.3.2 Timelock

No issues found.

2 Findings

2.1 MasterChef

The Masterchef is a fork of PancakeSwap's Masterchef but with the `migrator` function removed, which as of late has been used maliciously to steal tokens staked in the Masterchef. We commend PolyKiwi on their decision in making this modification of the Masterchef. PolyKiwi has also imposed an upper limit of 4.2% on deposit fees in the MasterChef, so the deposit fee cannot be set to a value higher than 4.2%.

2.1.2 Privileged Roles

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

- `add`
- `set`
- `setFeeAddress`
- `updateEmissionRate`
- `toggleReferrals`

2.1.3 Issues & Recommendations

Issue #01	**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, this will result in significant excessive rewards. Due to the way the Masterchef handles rewards, 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 issue is further amplified on Masterchefs like this one with a referral mechanism, since tokens can be minted directly. This flaw of the Masterchef has recently been exploited on a significant number of projects, all of which their native tokens went to $0 afterwards because the exploit resulted in a large number of native tokens being minted and dumped. 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	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);`
	
	Note that by using this method, you can also add the specific transfer tax logic for the native token if you so wish.

Resolution	RESOLVED The before-after method has been implemented.

Issue #02	`emergencyWithdraw` transfers no funds
Severity	HIGH SEVERITY
Description	In the `emergencyWithdraw` function, `user.amount` is set to 0 before the transfer function, with the end result being that no funds are transferred when this function is called as the user balance has been removed.

Recommendation	Consider first caching `user.amount` as follows :
	`uint256 amount = user.amount;`
	`user.amount = 0;`
	`user.rewardDebt = 0;`
	`pool.lpToken.safeTransfer(msg.sender, amount);`

Resolution	RESOLVED `user.amount` is transferred out before being set to zero. Note that this does not implement the Check-Effects Interaction pattern.

Issue #03	User balances does not properly record deposits if there are no deposit fees
Severity	HIGH SEVERITY
Location	Line 196 `user.amount.add(_amount);`
Description	This does not correctly add `_amount` to the current balance of the user.
Recommendation	This can be resolved by correcting to the following : `user.amount = user.amount.add(_amount);`
Resolution	RESOLVED The client has implemented the recommendation.

Issue #04	`payReferral` function can fail if there are insufficient tokens
Severity	MEDIUM SEVERITY
Description	The `payReferral` function transfers tokens from the `feeAddress` to the referral address. This is contingent on the `feeAddress` actually having sufficient tokens to pay on each transfer, which may fail should there be insufficient tokens or even zero tokens. This may happen if, for example, the `feeAddress` is controlled by the project team and can freely withdraw tokens, or if the referral payment amount exceeds the balance in the feeAddress.

Recommendation	Consider minting referral rewards rather than transferring them, as is the mechanism in most Pantherswap forks. This ensures that referral rewards will always be paid out, and can be implemented as such:
	`function payReferral (address referral, uint amount) internal {`
	`  if (referral != address(0) && referralStatus == true) {`
	`  amount = amount.div(40); // 2.5%`
	`  kiwi.mint(referral, amount);`
	`  emit ReferralPayment(referral, address(this), amount);`
	`  }`
	`}`

Resolution	RESOLVED Kiwi tokens are now minted to referrals.

Issue #05	Duplicated pools may be added to the Masterchef
Severity	MEDIUM SEVERITY
Description	The add function allows for duplicate pools to be added, which would lead to dilution of emission rewards to stakers.
Recommendation	The addition of a modifier that checks for duplicate pools could help prevent this incident from occurring.

```
mapping(IBEP20 => bool) public poolExistence;
modifier nonDuplicated(IBEP20 _lpToken) {
    require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated");
    _;
}

// Add a new lp to the pool. Can only be called by the owner.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool
_withUpdate) external onlyOwner nonDuplicated(_lpToken) {
    require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolExistence[_lpToken] = true;
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accH2OPerShare: 0,
        depositFeeBP: _depositFeeBP
    }));
    updateStakingPool();
}
```

Alternatively, you could account for this by adding in an `lpSupply` variable under `poolInfo`. This has the benefit of accurately accounting for deposits in the Masterchef.

Resolution	RESOLVED `nonDuplicate` modifier and `lpSupply` variable have been added.

Issue #06	Setting `feeAddress` to the zero address will break most functionality
Severity	LOW SEVERITY
Description	In the `deposit` function, the deposit fees are paid to the `feeAddress`. Should this be set to the zero address, then transfers will fail and thus break the `deposit` function.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like `require(_feeAddress != address(0), "!nonzero");` to the configuration function.
Resolution	RESOLVED

Issue #07	The `pendingKiwi` function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	In the `pendingKiwi` 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 `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so: `if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	RESOLVED

Issue #08	**Adding an EOA or a 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, and will fail if the token is not an actual token contract address.
Recommendation	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED

Issue #09	**`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	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_kiwiPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED A maximum emission rate of 0.0002 tokens per block has been set.

Issue #10	**Native token and associated comments still reference CAKE (from PancakeSwap)**
Severity	INFORMATIONAL
Description	Throughout the contract, the native token is still named and referred to as CAKE. As the project is called PolyKiwi, we are unsure if the native token is truly meant to retain the CAKE name, or if there is a specific name they would like their token to be known.
Recommendation	Consider renaming the CAKE token to the project's intended native token name, as well as making the necessary adjustments to the associated comments and functions. If CAKE is truly the project's native token, then this is a non-issue.
Resolution	PARTIALLY RESOLVED Comments still reference CAKE but functions and variables have been renamed to Kiwi.

Issue #11	**`BONUS_MULTIPLIER` looks to be redundant**
Severity	INFORMATIONAL
Description	The constant variable `BONUS_MULTIPLIER` does not contain any extra information since it is constant and cannot be changed from `1`, and thus would not have any impact or contribution on the `getMultiplier` function.
Recommendation	Consider removing `BONUS_MULTIPLIER` and associated comments.
Resolution	RESOLVED

Issue #12	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `accKiwiPerShare` is based on the `lpSupply` variable. `pool.accKiwiPerShare = pool.accKiwiPerShare.add(kiwiReward.mul(1e12).div(lpSupply));` However, if this `lpSupply` becomes a severely large value, precision errors may occur due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to 1e18 across the entire contract.
Resolution	RESOLVED

Issue #13	Various unused comments remain in the contract
Severity	INFORMATIONAL

Description	There are various comments in the contract, including removed functions, that can be removed to make the contract more readable by third-party reviewers.
	
	These include but are not limited to :
	`//function updateMultiplier(uint256 multiplierNumber) public onlyOwner {`
	`//    BONUS_MULTIPLIER = multiplierNumber;`
	`// cake.mint(devaddr, cakeReward.div(10));`

Recommendation	Consider removing these comments.
Resolution	RESOLVED

Issue #14	`kiwi` and `startBlock` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` 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 `immutable`.
Resolution	RESOLVED

Issue #15	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	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits.
Resolution	RESOLVED

Issue #16	**`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `setFeeAddress`, `updateEmissionRate`, `toggleReferrals` can be made `external`**
Severity	INFORMATIONAL
Description	The above 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.
Recommendation	Consider making these functions `external`.
Resolution	RESOLVED

Issue #17	**Lack of events for `add`, `set`, `setFeeAddress`, `updateEmissionRate`, `toggleReferrals`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for `add`, `set`, `setFeeAddress`, `updateEmissionRate`, `toggleReferrals`.
Resolution	RESOLVED

2.2 Timelock

The Timelock contract is a clean fork of Compounder 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	-	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	24 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.

2.2.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY