PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For PolyQuail

03 September 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 PolyQuail Finance 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	PolyQuail Finance
URL	https://www.polyquail.finance
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`MasterChef`	`0x439E9BE4618bfC5Ebe9B7357d848F65D24a50dDE`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2		
Medium	1	1		
Low	7	7		
Informational	11	11		
Total	**21**	**21**	**0**	**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	Contract uses raw addition and subtraction	RESOLVED
03	MEDIUM	Deposits may fail if previousMcAddrs is mis-specified	RESOLVED
04	LOW	The `pendingCHK` function will revert if totalAllocPoint is zero	RESOLVED
05	LOW	Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`	RESOLVED
06	LOW	`accCHKPerShare` still increases despite minting stopping	RESOLVED
07	LOW	`massUpdatePools` fails when too many pools are added	RESOLVED
08	LOW	`deposit` function may revert if `depositFeeBP` is zero	RESOLVED
09	LOW	`updateEmissionRate` has no maximum safeguard	RESOLVED
10	LOW	Setting `devAddress` to the zero address will break `updatePool`	RESOLVED
11	INFO	`chk` can be made `immutable`	RESOLVED
12	INFO	Unused functions and variables can be deleted	RESOLVED
13	INFO	Pools use the contract balance to figure out the total deposits	RESOLVED
14	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
15	INFO	`try/catch` statements should print error messages and events	RESOLVED
16	INFO	`getDepositFeeDiscountBP` and `getDepositFeeBP` should use uint256	RESOLVED
17	INFO	Inaccurate comment in `getDepositFeeDiscountBP`	RESOLVED
18	INFO	`updateEmissionIfNeeded` is redundant	RESOLVED
19	INFO	`dev` function can be renamed to `setDevAddress`	RESOLVED
20	INFO	`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `dev`, `setFeeAddress`, `updateEmissionRate` functions can be made `external`	RESOLVED
21	INFO	Lack of events for `add`, `set`, `updateEmissionIfNeeded`	RESOLVED

2 Findings

2.1 MasterChef

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

The main other extension this contract has is that users who had deposited and harvested above a certain amount in previous Masterchefs of the protocol (prior layers) receive discounted deposit fees. Currently this is 0.25% per previous Masterchef.

2.1.1 Privileged Roles

The ownership of the token contract should be transferred to the Masterchef. The following functions can be called by the owner of the contract:

- `add`
- `set`
- `dev`
- `setFeeAddress`
- `updateEmissionRate`
- `updateEmissionIfNeeded`
- `updateStartBlock`

2.1.2 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);
```

Resolution	RESOLVED

Issue #02	Contract uses raw addition and subtraction
Severity	HIGH SEVERITY

Location	Line 1486
	`totalDiscount += 25;`
	
	Line 1500
	`return pool.depositFeeBP - getDepositFeeDiscountBP(_user);`

Description	The `getDepositFeeBP` and `getDepositFeeDiscountBP` functions use either raw subtraction or addition. The risk of underflows in the `getDepositFeeBP` function is high because if `pool.depositFeeBP` is subtracted by a higher number `getDepositFeeDiscountBP(_user)`, this will inevitably result in underflows.
Recommendation	Consider using SafeMath's add and sub instead.
Comments	RESOLVED

Issue #03	**Deposits may fail if `previousMcAddrs` is mis-specified**
Severity	MEDIUM SEVERITY
Description	Users are given a 0.25% discount on their deposit fees if they harvest above a certain threshold on previous Masterchefs. Up to 4 previous Masterchefs are added in the constructor, with their respective minimum harvest amounts. If any one of these previous Masterchefs is mis-specified, such as having a typographical error in the address, adding in a non-Masterchef contract, or adding in an EOA, then the `getDepositFeeBP` function will fail. The ultimate result is that the `deposit` function will also fail as it attempts to call `getDepositFeeBP`.
Recommendation	This can be resolved by ensuring that the previous Masterchefs that will be initialized in the constructor are valid Masterchefs, and to thoroughly ensure that the mechanics of this function do not revert in any form via extensive testing of edge cases.
Resolution	RESOLVED The previous Masterchefs will query `poolLength`, and will fail if it is an invalid Masterchef contract.

Issue #04	The `pendingCHK` function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	In the `pendingCHK` 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 #05	Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`
Severity	LOW SEVERITY
Description	`updateEmissionRate` 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 #06	accCHKPerShare still increases despite minting stopping
Severity	LOW SEVERITY
Description	The `accCHKPerShare` will still continue to increase despite the `mint` functions failing to execute in the `try/catch` statements in `updatePool` function.

Recommendation	Consider setting `chkReward` to zero in the `catch` statement. This ensures that `accCHKPerShare` will not continue to increase after farming emissions are over.
	
	`try chk.mint(devaddr, chkReward.div(10)) {`
	`} catch (bytes memory reason) {`
	`  chkReward = 0;`
	`  emit CHKMintError(reason);`
	`}`
	
	`try chk.mint(address(this), chkReward) {`
	`} catch (bytes memory reason) {`
	`  chkReward = 0;`
	`  emit CHKMintError(reason);`
	`}`

Resolution	RESOLVED

Issue #07	`massUpdatePools` fails when too many pools are added
Severity	LOW SEVERITY
Description	In the event that the Masterchef has over 50 pools added, `massUpdatePools` will fail. This will result in rewards potentially being miscalculated because any changes to `totalAllocPoints` is not reflected, and it would thus be very costly and time-consuming to manually call `updatePools` on each `pid`. This issue is marked low severity as it only affects the `add`, `set`, and `updateEmissionRate` functions in the rare event that the Masterchef does indeed have over 50 pools.
Recommendation	If there is ever the case of the Masterchef having so many pools, ensure that the `updatePools` function is called manually should there be any changes to `allocPoints` that may arise due to `add`, `set` or `updateEmissionRate`. This issue can then be marked as resolved upon acknowledgement of this issue.
Resolution	RESOLVED The client has acknowledged the issue and will manually call `updatePools` if need be.

Issue #08	**deposit function may revert if depositFeeBP is zero**
Severity	LOW SEVERITY

Location	Lines 1529-1533
	`if (pool.depositFeeBP > 0) {`
	`    uint256 depositFeeBP = getDepositFeeBP(_pid, msg.sender);`
	`    uint256 depositFee = _amount.mul(depositFeeBP).div(10000);`
	`    pool.lpToken.safeTransfer(feeAddress, depositFee);`
	`    user.amount = user.amount.add(_amount).sub(depositFee);`

Description	If `getDepositFeeBP` returns 0, then it will attempt to transfer 0 tokens to the `feeAddress`. Certain tokens will revert transfers if sending an amount equal to zero.

Recommendation	Consider adding in a further non-zero `depositFeeBP` check, like so:
	`if (pool.depositFeeBP > 0) {`
	`    uint256 depositFeeBP = getDepositFeeBP(_pid, msg.sender);`
	`    if (depositFeeBP > 0) {`
	`        uint256 depositFee = _amount.mul(depositFeeBP).div(10000);`
	`        pool.lpToken.safeTransfer(feeAddress, depositFee);`
	`        user.amount = user.amount.add(_amount).sub(depositFee);`
	`    } else {`
	`        user.amount = user.amount.add(_amount);`
	`    }`
	`} else {`
	`    user.amount = user.amount.add(_amount);`

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(_CHKPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED There is now an upper limit of 1 token per block.

Issue #10	Setting `devAddress` to the zero address will break `updatePool`
Severity	LOW SEVERITY
Description	`updatePool` function will mint tokens to the `devAddress`, but minting will revert if it is made to the zero address. This will thus cause deposits and withdrawals to fail.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following: `require(_devAddress != address(0), "!nonzero");` to the `setDevAddress` function.
Resolution	RESOLVED

Issue #11	chk 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 chk explicitly `immutable`.
Comments	RESOLVED

Issue #12	Unused functions and variables can be deleted
Severity	INFORMATIONAL
Description	There are plenty of instances throughout the contract whereby functions have been commented out, and can thus be deleted. These include: Lines 1262-1266 Lines 1282-1287 Lines 1544-1593
Recommendation	Consider deleting unused comments and functions to reduce the length of the contract for third-party reviewers.
Comments	RESOLVED

Issue #13	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.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. Each `lpToken.balanceOf(address(this))` query can then be replaced with this `lpSupply` as well.
Comments	RESOLVED

Issue #14	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `pool.accCHKPerShare` is based on the `lpSupply` variable. `pool.accCHKPerShare = pool.accCHKPerShare.add(chkReward.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 #15	`try/catch` statements should print error messages and events
Severity	INFORMATIONAL
Description	It is recommended, for debugging or post-mortem requirements, for the `try/catch` statements to print error messages as detailed in this blog post. It should also emit events when an error does occur.
Recommendation	Consider including errors raised and emitting events in the `try/catch` statements.
Resolution	RESOLVED

Issue #16	`getDepositFeeDiscountBP` and `getDepositFeeBP` should use `uint256`
Severity	INFORMATIONAL
Description	It is slightly more gas intensive to use `uint16` rather than `uint256` as Solidity has to downscale from the latter to the former.
Recommendation	Consider using `uint256` instead.
Resolution	RESOLVED

Issue #17	Inaccurate comment in `getDepositFeeDiscountBP`
Severity	INFORMATIONAL
Description	The comment states that there is 50 bps of discounts for each Masterchef harvest exceeding the minimum threshold. The actual discount is, however, 25 bps.
Recommendation	Consider either correcting the `totalDiscount` figure to 50 bps or updating the comment to 25 bps.
Resolution	RESOLVED

Issue #18	`updateEmissionIfNeeded` is redundant
Severity	INFORMATIONAL
Description	Currently, when this function is called, it will set emission rates to zero. However, this function is not needed because the `try/catch` statement in `updatePool` will stop token minting when maximum supply has been reached in the underlying token contract itself.
Recommendation	Consider removing this function.
Resolution	RESOLVED

Issue #19	**dev function can be renamed to `setDevAddress`**
Severity	INFORMATIONAL
Description	The name of the function is a variable, and should be renamed to `setDevAddress` to reduce possible confusion.
Recommendation	Consider renaming this function to `setDevAddress`.
Resolution	RESOLVED

Issue #20	**`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `dev`, `setFeeAddress` and `updateEmissionRate` functions 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 #21	**Lack of events for `add`, `set`, `updateEmissionIfNeeded`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY