



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For PolyWhale SmartChef

21 September 2021



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 SmartChef	6
2 Findings	7
2.1 SmartChef	7
2.1.1 Privileged Roles	7
2.1.2 Issues & Recommendations	8

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 advocacy 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 PolyWhale Finance's SmartChef contract. 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	PolyWhale SmartChef
URL	https://polywhale.finance
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
GulfPools	https://github.com/Alsatoshi/gulf_pools-smart-contract	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	1	0	-	1
● Medium	2	2	-	-
● Low	2	2	-	-
● Informational	4	4	-	-
Total	9	8	-	1

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 SmartChef

ID	Severity	Summary	Status
01	MEDIUM	Severely excessive rewards issue when a token with a transfer tax is added	UNRESOLVED
02	MEDIUM	Rewards can be stopped by owner at any time	RESOLVED
03	MEDIUM	deposit, withdraw and emergencyWithdraw functions is are vulnerable to reentrancy	RESOLVED
04	LOW	burnMultiplier can be set to any amount	RESOLVED
05	LOW	Lack of events for adjustBlockEnd and stopRewards	RESOLVED
06	INFO	stopReward, adjustBlockEnd, emergencyWithdraw, withdraw and deposit can be made external	RESOLVED
07	INFO	krill, rewardToken, rewardPerBlock and maxDeposit can be made immutable	RESOLVED
08	INFO	Contract should use safeMath for arithmetic operations	RESOLVED
09	INFO	Deposits may exceed maxDeposit	RESOLVED

2 Findings

2.1 SmartChef

The PolyWhale SmartChef contract is a fork of Pancakeswap's SmartChef, where users can stake Krill tokens to earn reward tokens. There is a specified start and end block, and the latter may be adjusted to a later period should more reward tokens be transferred in to this contract.



2.1.1 Privileged Roles



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



- `stopReward`
- `adjustBlockEnd`



2.1.2 Issues & Recommendations


Issue #01	Severely excessive rewards issue when a token with a transfer tax is added
Severity	 HIGH SEVERITY
Description	<p>If the Krill token contains transfer taxes, this may result in significantly excessive rewards being paid out as the balance recorded does not accurately match the balance in the Masterchef. This flaw has recently been exploited on a significant number of projects, the most recent of which was PolyYeld. In all cases, their native tokens went to \$0 because the exploit resulted in an egregiously large number of tokens being minted and dumped.</p> <p>This issue was also present in SushiSwap (the original Masterchef) and is present in pretty much every Masterchef since. Since the Masterchefs were never meant to have any tokens but LP tokens, this issue did not manifest previously but has become a problem to projects who have started forking it for usage with less standard tokens.</p>
Recommendation	<p>Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:</p> <pre>uint256 balanceBefore = pool.lpToken.balanceOf(address(this)); pool.lpToken.transferFrom(msg.sender, address(this), _amount); _amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);</pre> <p>Note that by using this method, you can also add the specific transfer tax logic for the native token if you so wish.</p>
Resolution	 UNRESOLVED
	The client did not implement the fix correctly.

Issue #02	Rewards can be stopped by owner at any time
Severity	 MEDIUM SEVERITY
Description	Calling the stopReward function will set multipliers to 0.
Recommendation	Remove this function if it is not needed. Otherwise, if there is a justifiable reason, then consider transferring ownership of the SmartChef contract to a minimum 1-day Timelock.
Resolution	 RESOLVED

Issue #03	deposit, withdraw and emergencyWithdraw functions is are vulnerable to reentrancy
Severity	 MEDIUM SEVERITY
Description	Adding a nonReentrant feature to the functions mentioned would prevent reentrancy attacks especially when ERC777 tokens are used, and arranging the order such that user balances are set to 0 before transfers (in the withdraw and emergencyWithdraw functions) are made would be a safe practice in line with the Check Effects Interactions pattern .
Recommendation	<p>To implement the Check Effects Interactions pattern for the emergencyWithdraw function, for example, consider the following:</p> <pre> function emergencyWithdraw() public nonReentrant{ PoolInfo storage pool = poolInfo[0]; UserInfo storage user = userInfo[msg.sender]; user.amount = 0; user.rewardDebt = 0; pool.lpToken.safeTransfer(address(msg.sender), user.amount); emit EmergencyWithdraw(msg.sender, user.amount); } </pre>
Resolution	 RESOLVED

Issue #04 burnMultiplier can be set to any amount

Severity

 LOW SEVERITY

Description

Currently, there is no upper limit to the value that burnMultiplier can be set. As user deposits are subtracted by the burnAmount, setting a reasonable limit and clearly informing your users of this limit would ensure that they are aware of any deductions from their deposit amount.

Recommendation

Consider setting a reasonable limit, generally in the range of 4-10%, in the constructor. This can be done like so:

```
require(_burnMultiplier <= 100, "Burn amount too high!");
```

Resolution

 RESOLVED

Issue #05 Lack of events for adjustBlockEnd and stopRewards

Severity

 INFORMATIONAL

Description

Functions that affect the status of sensitive variables should emit events as notifications.

Recommendation


Consider adding an event for these functions.

Resolution

 RESOLVED

Issue #06 **stopReward, adjustBlockEnd, emergencyWithdraw, withdraw and deposit can be made external**

Severity

 INFORMATIONAL

Description

Consider changing the above functions 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 #07 **krill, rewardToken, rewardPerBlock and maxDeposit 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.

Recommendation

Consider making the above variables explicitly `immutable`.

Resolution

 RESOLVED

Issue #08**Contract should use SafeMath for arithmetic operations****Severity** INFORMATIONAL**Description**

Currently, the SmartChef contract uses Solidity version 0.6.12, which does not have SafeMath automatically implemented (this is present in versions 0.8.0 and later). As such, arithmetic operations that use raw subtraction and addition may be subject to underflows and overflows respectively.

There are several instances in the contract where raw additions and subtractions are used:

Line 671

```
bonusEndBlock = block.number + totalLeft.div(rewardPerBlock);
```

Line 732

```
pool.lpToken.safeTransferFrom(address(msg.sender), address(this),  
_amount - burnAmount);
```

Line 736

```
user.amount = user.amount.add(_amount - burnAmount);
```

Recommendation

Consider using SafeMath's add and sub instead, or upgrading to Solidity versions 0.8.0 and higher.

Resolution RESOLVED

Severity

 INFORMATIONAL

Description

The maxDeposit variable may be exceeded if, for example, the amount of Krill tokens in the SmartChef contract is just under the stated deposit hard-cap, and a user deposits tokens equalling to `_amount` that results in maxDeposit being exceeded. To visualize this situation, consider the following :

maxDeposit = 250,000 tokens

pool.lpToken.balanceOf(address(this)) = 249,950 tokens

_amount = 10,000 tokens

The result of this deposit is that there are now 259,950 tokens in the SmartChef contract.

Recommendation

Consider adjusting the require statement to account for the `_amount` that is to be deposited by the user, like so:

```
require(pool.lpToken.balanceOf(address(this)).add(_amount) <=
maxDeposit, "Deposit limit reached!!");
```

Resolution

 RESOLVED



PALADIN
BLOCKCHAIN SECURITY