PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For MorpheusSwap Finance

21 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 the MorpheusSwap Finance on the Fantom Opera 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

MorpheusSwap is a yield farm in Fantom chain in which contracts are forked from PolyPup Ball's post-audited contracts in Polygon.

Project Name	MorpheusSwap
URL	https://morpheusswap.finance/
Platform	Fantom Opera
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`MasterChef`	`0xc7dad2e953Dc7b11474151134737A007049f576E`	MATCH
`MorpheusToken`	`0x0789fF5bA37f72ABC4D561D00648acaDC897b32d`	MATCH
`Timelock`	`0x77e4b42c3d788735bc27ad7f494362b07ccd9f04`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	2	-	-	2
Low	2	1	-	1
Informational	8	-	-	8
Total	**12**	**1**	**-**	**11**

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	MEDIUM	Setting `devaddr` to the zero address will break deposit and withdrawals	ACKNOWLEDGED
02	MEDIUM	`deposit` function is susceptible to underflow	ACKNOWLEDGED
03	LOW	`updateEmissionRate` has no maximum safeguard	ACKNOWLEDGED
04	INFO	`BONUS_MULTIPLIER` is redundant	ACKNOWLEDGED
05	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
06	INFO	`morph` can be made immutable	ACKNOWLEDGED
07	INFO	`_withUpdate` input can be automated in the set function to eliminate manual intervention	ACKNOWLEDGED
08	INFO	`renounceOwnership` and `transferOwnership` can be made `external`	ACKNOWLEDGED

1.3.2 MorpheusToken

ID	Severity	Summary	Status
09	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
10	INFO	Governance functionality is broken	ACKNOWLEDGED
11	INFO	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED
12	INFO	Comment is still referring to VICTs	ACKNOWLEDGED

1.3.6 Timelock

No issues found.

2 Findings

2.1 MasterChef

The MasterChef contract is a fork of PolyPup Ball's Masterchef, which does not have the `migrator` function. The Masterchef also uses `block.timestamp` instead of `block.number` when calculating the pending rewards.

2.1.1 Contract Dependencies

- `Ownable`
- `ReentrancyGuard`
- `SafeMath`
- `SafeBEP20`

2.1.2 Privileged Roles

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

- `add`
- `set`
- `setDevAddress`
- `setFeeAddress`
- `updateEmissionRate`
- `updateStartTimestamp`
- `renounceOwnership`
- `transferOwnership`

2.1.3 Issues & Recommendations

Issue #01	**Setting devaddr to the zero address will break deposit and withdrawals**
Severity	MEDIUM SEVERITY

Location	Lines 1520-1524
	`function setDevAddress(address _devaddr) external {`
	`  require(msg.sender == devaddr, "dev: wut?");`
	`  devaddr = _devaddr;`
	`  emit SetDevAddress(msg.sender, _devaddr);`
	`}`

Description	Within the token contract, minting tokens to the zero address will revert transactions. Deposits and withdrawals will break if the devaddr is ever set to the zero address due to the `updatePool` function. Harvesting will fail as well.
Recommendation	To prevent this from happening by accident, and to limit governance risks, consider adding a requirement to the `devaddr` that the new address should not be equal to zero.
Resolution	ACKNOWLEDGED The client has acknowledged this issue and will ensure that the `devaddr` will never be set to the zero address.

Issue #02	**deposit function is susceptible to underflow**
Severity	MEDIUM SEVERITY
Location	Line 1455 `_amount = pool.lpToken.balanceOf(address(this)) - balanceBefore;`
Description	There is raw subtraction in the deposit function. As this contract uses Solidity version 0.6.12, it may be susceptible to integer underflows.
Recommendation	Consider upgrading to Solidity version 0.8.0 or higher, or using SafeMath's sub rather than raw subtraction.
Resolution	ACKNOWLEDGED The client mentioned that they have taken measures to accurately calculate before and after balances as this issue would only materialize if the before is larger than after, and this seldom occurs.

Issue #03	**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(_MorphPerSec <= MAX_EMISSION_RATE,"Too high");`
Resolution	ACKNOWLEDGED The client acknowledges this issue and mentioned that this function will either not be used or used rarely, and when used, utmost care will be taken to correctly and safely set the emission rate.

Issue #04	**BONUS_MULTIPLIER is redundant**
Severity	INFORMATIONAL
Description	The constant variable `BONUS_MULTIPLIER` does not look to be used in the Masterchef contract and can thus be removed.
Recommendation	Consider removing `BONUS_MULTIPLIER` and associated comments.
Resolution	ACKNOWLEDGED

Issue #05	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `accMorphPerShare` is based on the `lpSupply` variable. `pool.accMorphPerShare = pool.accMorphPerShare.add(morphReward.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	ACKNOWLEDGED The client has acknowledged this issue and has no plans to add any meme tokens. All tokens added will also be reviewed carefully.

Issue #06	`morph` 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 `morph` explicitly `immutable`.
Resolution	ACKNOWLEDGED

Issue #07	`_withUpdate` input can be automated in the set function to eliminate manual intervention
Severity	INFORMATIONAL
Description	Manual inputs are prone to human error and eliminating them can bring about efficiency with an unnoticeable gas cost.
Recommendation	For the set function, consider comparing the current values in the poolInfo with the input and should the current value not be equal with the input value, the function can then call massUpdatePools( ).
Resolution	ACKNOWLEDGED

Issue #08	`renounceOwnership` and `transferOwnership` can be made `external`
Severity	INFORMATIONAL
Description	Public functions that are never called by the contract should be declared external to save gas.
Recommendation	Use the `external` attribute for functions never called from the contract.
Resolution	ACKNOWLEDGED

2.2 MorpheusToken

The MorpheusToken contract is an ERC-20 implementation which will be used as the main reward token for the Masterchef.

2.2.1 Token Overview

Address	0x0789fF5bA37f72ABC4D561D00648acaDC897b32d
Token Supply	2 million (emissions will be set to 0 manually once max supply has been reached)
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

2.2.2 Contract Dependencies

- `Ownable`
- `Context`
- `BEP20`
- `IBEP20`
- `SafeMath`
- `Address`

2.2.3 Privileged Roles

The following are functions of the MorpheusToken that is callable by the owner:

- `mint`
- `renounceOwnership`
- `transferOwnership`

2.2.4 Issues & Recommendations

Issue #09	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint tokens for dumping. As token ownership has already been transferred to the Masterchef, the main risk that remains is whether any tokens were pre-minted to the project owner prior to transferring ownership.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED The client has minted 900 tokens for initial liquidity, with 100 minted to cover initial partnerships. No more tokens were manually minted and ownership has been transferred to the Masterchef.

Issue #10	Governance functionality is broken
Severity	INFORMATIONAL
Description	Although there is YAM-related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.
Recommendation	The broken delegation-related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, used by many of the larger projects.
Resolution	ACKNOWLEDGED

Issue #11	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation	The simplest solution would be to just remove the voting-related mechanism in the token contract, and relying on snapshot should there ever be a need for voting in the future. Alternatively, Consider adding the desired message sender in the `structhash` and requiring this desired sender to be equal to `msg.sender`. This reduces the problem to having the message sender be able to frontrun you, which is okay if it is a reviewed contract.
Resolution	ACKNOWLEDGED

Issue #12	Comment is still referring to VICTs
Severity	INFORMATIONAL
Location	Line 1053 `uint256 delegatorBalance = balanceOf(delegator); // balance of underlying VICTs (not scaled);`
Description	There is a comment in the code that refers to VICTs.
Recommendation	Consider correcting if unintended to be in line with the token name.
Resolution	ACKNOWLEDGED

2.3 Timelock

The Timelock contract is a clean fork of Compound Finance's timelock that was upgraded to Solidity version 0.6.12 and slight changes to the delays. 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	7 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	2 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.
Maximum Delay	30 days	The maximum delay indicates the highest value that the `delay` can be set.
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.


PALADIN
BLOCKCHAIN SECURITY