PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Farmers Only

02 October 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 FarmersOnly on the Avalanche 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	Farmers Only
URL	https://farmersonly.farm
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
MasterChef	0x27271ECd985F1B666e51209B033d93ddC5a36076	✓ MATCH
CornCoin	0xFcA54c64BC44ce2E72d621B6Ed34981e53B66CaE	✓ MATCH
Multicall	Only used on the frontend	✓ MATCH
LiqLocker	0xAfbD20B5E79fAdBc0c0B9AbA64cBb5d27D995bFD	✓ MATCH
Timelock	0xc14BFa11c072d5e5735092CEe376Be905B9D7359	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	1	-	1	-
🟠 Medium	1	-	-	1
🟡 Low	4	-	-	4
🟣 Informational	11	-	-	11
Total	**17**	**-**	**1**	**16**

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	Harvest rewards are sent twice to the user: one iteration circumvents the harvest lock	PARTIAL
02	MEDIUM	Gov privilege: Harvest interval can be set up to 600 days preventing harvests	ACKNOWLEDGED
03	LOW	`updateEmissionRate` has no maximum safeguard	ACKNOWLEDGED
04	LOW	`poolExistence` mapping not actively used to prevent addition of the same `lpToken` twice	ACKNOWLEDGED
05	LOW	Unsafe subtraction in `deposit` function	ACKNOWLEDGED
06	INFO	Usage of implicit `return` variables	ACKNOWLEDGED
07	INFO	Typographical errors in event capitalization	ACKNOWLEDGED
08	INFO	Internal `canHarvest` could break third party tools	ACKNOWLEDGED
09	INFO	Unecessary integer arithmetic within `userPoolLocked`	ACKNOWLEDGED
10	INFO	Unnecessary casting of `msg.sender` to `address`	ACKNOWLEDGED
11	INFO	Unused variable `endTime`	ACKNOWLEDGED

1.3.2 CornCoin

ID	Severity	Summary	Status
12	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED
13	INFO	Gas Optimizations: Unnecessary usage of `MAX_CAP` variable to keep track of total supply since can be used	ACKNOWLEDGED

1.3.3 Multicall

No issues found.

1.3.4 LiqLocker

ID	Severity	Summary	Status
14	INFO	Usage of transfer in contrast to `safeTransfer` might prevent withdrawals on tokens which are not implemented correctly	ACKNOWLEDGED
15	INFO	`farmersOnlyDev` can be made `constant`	ACKNOWLEDGED
16	INFO	`unlockTimestamp` can be made `immutable`	ACKNOWLEDGED
17	INFO	Lack of event for the `withdraw` function	ACKNOWLEDGED

1.3.5 Timelock

No issues found.

2 Findings

2.1 MasterChef

The Masterchef is a fork of Goose Finance's Masterchef with inspiration drawn from other projects like PantherSwap. A notable feature of forking the latter is the removal of the `migrator` function, which as of late has been used maliciously to steal user's tokens. In addition, in comparison to Goose Finance, FarmersOnly has limited the deposit fee to at most 4%. We commend FarmersOnly 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 most notable features of this version of the MasterChef are the harvest lockups and deposit fee limitation. The initial emission rate is 0.0035 tokens per second.

2.1.1 Privileged Roles

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

- `add`
- `set`
- `setDevAddress`
- `setFeeAddress`
- `updateEmissionRate`
- `updateStartTime`

2.1.2 Issues & Recommendations

Issue #01	**Harvest rewards are sent twice to the user: one iteration circumvents the harvest lock**
Severity	● HIGH SEVERITY
Description	In the current implementation of FarmersOnly, both a Panther-like and Goose-like reward mechanism are included. Since we could not find any information on the frontend with regards to the Goose-like reward mechanism, we assume it is included by accident. Currently, whenever a reward is harvested, it is both locked up and distributed to the user resulting in an eventual double reward for users.

Recommendation	Consider removing the goose-like reward distribution within `deposit` and `withdraw`:
	
	`if (user.amount > 0) {`
	`  uint256 pending =`
	`user.amount.mul(pool.accCornPerShare).div(1e18).sub(user.rewardDe`
	`bt);`
	`  if (pending > 0) {`
	`    safeCornTransfer(msg.sender, pending);`
	`  }`
	`}`

Resolution	● PARTIALLY RESOLVED As the client did not want to redeploy due to their launch being imminent, we worked together with the client to find a way to allow them to mitigate this issue without redeploying. First of all, there were two ways the project could deal with this issue without redeploying: 1) they could either claim the feature and be the first farm to combine both Goose and Panther features, or 2) they could set the harvest locks to zero which would result in them becoming a Goose-like Masterchef with twice the emission rate. The client opted for the latter (no harvest locks). But there was a very important issue remaining: while users would receive twice their allocated rewards, the Masterchef would only mint them once. This would result in the Masterchef quickly being drained of native tokens and the native pool no longer being able to withdraw. As a way to mitigate this issue, the client added a dummy pool at pid 13 which mints half of the emission rate but is never harvestable. These tokens can then be used as rewards for the other pools. It should be noted that the admin should always maintain half of the allocation points on this pool and call `updatePool(13)` sufficiently frequently. This issue is only marked as partially resolved since the code still has this vulnerability. Due to the mitigation, the consequences will not be present as long as the dummy pool maintains 50% of the allocation points and updatePool is called periodically on it. It is thus properly mitigated within the active deployment of the code but we leave this issue partially open to prevent this Masterchef from being blindly copied by third-party forks.

Issue #02	Gov privilege: Harvest interval can be set up to 600 days preventing harvests
Severity	● MEDIUM SEVERITY

Location	Line 1083
	`uint256 public constant MAXIMUM_HARVEST_INTERVAL = 4 hours;`
	
	Line 1185
	`poolInfo[_pid].harvestInterval = _harvestInterval * 1 hours;`

Description	The code allows a harvest interval of at most 4 * 1 hour * 1 hour second to be set by the governance. We expect that the goal was to limit this interval to 4 * 1 hour however. Since the harvest interval is specified in seconds, the harvest interval can be set to 3600 times higher than what is intended. Note that for users this is not really a larger risk than the standard governance privileges. Even when this is resolved, in any Masterchef, the admin can simply disable rewards on a pool by setting the pool `allocPoints` to zero. This issue has thus been marked as medium risk not because of the governance privilege, but because the intention of the code was wrongly implemented.
Recommendation	Consider not multiplying by 1 hours in the `harvestInterval` setting.
Resolution	● ACKNOWLEDGED

Issue #03	**`updateEmissionRate` has no maximum safeguard**
Severity	● LOW SEVERITY
Description	Projects sometimes 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(_cornPerSecond <= MAX_EMISSION_RATE,"Too high");`
Resolution	● ACKNOWLEDGED

Issue #04	**`poolExistence` mapping not actively used to prevent addition of the same `lpToken` twice**
Severity	● LOW SEVERITY
Description	The code contains a `poolExistence` mapping that keeps track of whether a token has been added as a pool. In addition, there is a `nonDuplicated` modifier that can be used to prevent a token from being added twice. However, this modifier is currently completely unused throughout the contract, which makes us suspect that this functionality is accidentally misimplemented.
Recommendation	Consider actively checking that `poolExistence` is `false` by adding the `nonDuplicated` modifier to the add function. `function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, uint256 _harvestInterval, bool _withUpdate) external onlyOwner nonDuplicated(_lpToken) {`
Resolution	● ACKNOWLEDGED

Issue #05	Unsafe subtraction in `deposit` function
Severity	● LOW SEVERITY
Location	Line 1267 `_amount = pool.lpToken.balanceOf(address(this)) - balanceBefore;`
Description	The `deposit` function contains a subtraction that is not protected against underflow. This could in theory result in the deposit amount being a very large number in case a weird token is added that decreases the contract balance after deposits. This issue is marked as low-risk since we know of no such tokens.
Recommendation	Consider using SafeMath's `sub` instead.
Resolution	● ACKNOWLEDGED

Issue #06	Usage of implicit return variables
Severity	● INFORMATIONAL

Location	Lines 1137-1139 (example)
	`function blockTimestamp() external view returns (uint time) {`
	`    time = block.timestamp;`
	`}`

Description	Using `return` variables and then implicitly returning without a `return` statement can be confusing for third-party reviewers as they need to double check that no globally scoped variable has been modified.
Recommendation	Consider following best-practice and returning variables explicitly. In the example the line of code would be replaced with the following code: `return block.timestamp;`
Resolution	● ACKNOWLEDGED

Issue #07	Typographical errors in event capitalization
Severity	INFORMATIONAL

Location	Lines 1141-1142
	`event addPool(uint256 indexed pid, address lpToken, uint256 allocPoint, uint256 depositFeeBP, uint256 harvestInterval);`
	`event setPool(uint256 indexed pid, address lpToken, uint256 allocPoint, uint256 depositFeeBP, uint256 harvestInterval);`

Description	Throughout the contract, events are capitalised; however for the `addPool` and `setPool` event, this capitalisation has been forgotten which might signal inconsistency to third-party reviewers.
Recommendation	Consider capitalising the two events in question to `AddPool` and `SetPool` as is best practice for events.
Resolution	ACKNOWLEDGED

Issue #08	Internal `canHarvest` could break third party tools
Severity	INFORMATIONAL
Location	Line 1242 `function canHarvest(uint256 _pid, address _user) internal view returns (bool) {`
Description	The `canHarvest` function is relatively popular among third-party tools that can interact with Panther-like Masterchefs. However, within this version of the contract, this function is marked as `internal`, making it impossible for these tools to call it.
Recommendation	Consider making the `canHarvest` function `public`. Furthermore, `user` can be of type `memory` to reduce gas usage.
Resolution	ACKNOWLEDGED

Issue #09	Unecessary integer arithmetic within `userPoolLocked`
Severity	● INFORMATIONAL

Location	Lines 1174-1178
	`function userPoolLockup(uint _pid, address _user) external view returns (int lock) {`
	`    UserInfo storage user = userInfo[_pid][_user];`
	`    lock = int(user.nextHarvestUntil) - int(block.timestamp);`
	`    if(lock < 0) lock = 0;`
	`}`

Description	It is best practice within solidity to avoid using integers that can go negative wherever they are not necessary, this is because third-party reviewers are often not as comfortable with them.

Recommendation	Consider simplifying the function to use simple `uint256`.
	`function userPoolLockup(uint _pid, address _user) external view returns (uint256) {`
	`    UserInfo storage user = userInfo[_pid][_user];`
	`    if (block.timestamp >= user.nextHarvestUntil) {`
	`        return 0;`
	`    }`
	`    return user.nextHarvestUntil- block.timestamp;`
	`}`

Resolution	● ACKNOWLEDGED

Issue #10	Unnecessary casting of `msg.sender` to `address`
Severity	● INFORMATIONAL
Description	Example: `address(msg.sender)` Throughout the contract, `msg.sender` is sometimes explicitly cast as an `address`. Since the type of this variable is already address, this is redundant.
Recommendation	Consider omitting these explicit casts.
Resolution	● ACKNOWLEDGED

Issue #11	Unused variable end`Time`
Severity	● INFORMATIONAL

Location	`Line 1126`
	`uint public endTime = 0;`

Description	Variables that are unused should be removed from the source code to reduce code complexity for third-party reviewers.
Recommendation	Consider removing the above variable.
Resolution	● ACKNOWLEDGED

2.2 CornCoin

The CornCoin is a simple ERC-20 token with a maximum supply cap of 7000 tokens. After the contract is created, ownership should be manually transferred by the admin to the MasterChef to allow the MasterChef to start minting tokens when farming starts. Once the total supply reaches 7000 tokens, minting will start reverting which should be handled within the Masterchef.

2.2.1 Token Overview

Address	0xFcA54c64BC44ce2E72d621B6Ed34981e53B66CaE
Token Supply	7,000
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

2.2.2 Privileged Roles

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

- `mint`

2.2.3 Issues & Recommendations

Issue #12	**`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 and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
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	● ACKNOWLEDGED

Issue #13	**Gas Optimizations: Unnecessary usage of `MAX_CAP` variable to keep track of total supply since `_totalSupply` can be used**
Severity	● INFORMATIONAL
Description	Traditionally, tokens with a maximum supply cap keep track of the amount of minted tokens through a variable `MAX_CAP`. However, since FarmersOnly also deducts this value when tokens are burned, it is in fact identical to the `_totalSupply` variable.
Recommendation	Consider whether the `MAX_CAP` is supposed to decrease again on burns (eg. the Masterchef can mint again after some tokens are burned). If this is desired behavior, `_totalSupply` can just be used; otherwise, consider removing the `MAX_CAP` adjustment from the `burn` function.
Resolution	● ACKNOWLEDGED

2.3 Multicall

The multicall contract is a clean fork of the standard Multicall contract by MakerDAO to provide more efficient frontend queries. It allows the frontend to query for multiple values in a single RPC call. This saves a lot of back-and-forth between the RPC and the frontend and improves the user experience.

It should be noted that if one of these subcalls fails, the whole multicall will fail. If partial failure is expected the client should consider MulticallV2.

2.3.1 Issues & Recommendations

No issues found.

2.4 FarmersOnlyLiqLocker

The FarmersOnlyLiqLocker is an LP locking contract that only allows the withdrawal of tokens after 1 year after deployment has passed.

For the deployed contract, tokens will be unlocked on Sun Oct 02 2022 08:23:40 GMT+00. Tokens can only be withdrawn by the dev wallet: 0xeE68753bD98d29D20C8768b05f90c95D66AEf1a8.

As of the time of publish of this audit, no tokens have been locked inside the locker.

2.4.1 Issues & Recommendations

Issue #14	Usage of transfer in contrast to `safeTransfer` might prevent withdrawals on tokens which are not implemented correctly
Severity	INFORMATIONAL
Description	A very small subset of tokens has wrongly implemented the ERC-20 specification in the sense that their `transfer` function does not return a boolean result. These tokens cannot be withdrawn through the FarmersOnlyLiqLocker. This issue is marked as informational as these days there are very few such tokens and neither LP tokens nor the CornCoin has this issue. We suspect it is unlikely that the dev will ever wish to lock such a token.
Recommendation	Consider using the OpenZeppelin SafeERC20.
Resolution	ACKNOWLEDGED

Issue #15	`farmersOnlyDev` can be made constant
Severity	INFORMATIONAL
Description	Variables that are never changed throughout the contract can be marked as constant for gas saving purposes.
Recommendation	Consider marking the above variables as constant.
Resolution	ACKNOWLEDGED

Issue #16	`unlockTimestamp` can be made `immutable`
Severity	● INFORMATIONAL
Description	Variables that are never changed throughout the contract but are defined within the constructor can be marked as immutable for gas saving purposes.
Recommendation	Consider marking the aforementioned variables as `immutable`.
Resolution	● ACKNOWLEDGED

Issue #17	Lack of event for the `withdraw` function
Severity	● INFORMATIONAL
Description	Important functions should emit events to keep a track record of when and how they have been called.
Recommendation	Consider adding events to the above function.
Resolution	● ACKNOWLEDGED

2.5 Timelock

The Timelock contract is a clean fork of Compound 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	8 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	8 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.

2.5.1 Issues & Recommendations

No issues found.

PALADIN

BLOCKCHAIN SECURITY