PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Cosmic Guild

03 July 2022

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.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Cosmic Guild on the BNB Smart Chain. 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	Cosmic Guild
URL	https://cosmicguild.io/
Network	BNB Smart Chain
Language	Solidity

The part of Cosmic Guild that is being audited is a yield farm. Users deposit their different cryptocurrency tokens, presumably the Cosmic Guild tokens, in the decentralized application and earn reward tokens in return.

General risks on yield farming projects include:

- Too much governance privilege where a single or group of addresses can control the flow of deposited funds in the project contracts (rug risk)
- Contract bugs that can lead to exploitation of miscalculated rewards or misplaced deposits/withdrawals.
- Too much complexity is involved in the flow of funds with auxiliary contracts, i.e., strategy contracts, which usually have risks of getting funds stuck or skimmed.

All in all, yield farms are generally a type of simple decentralized finance protocol and relatively safe compared to other protocol types as long as the above risks are mitigated.

Below, the audit will cover issues found in the Cosmic Guild protocol and provide recommendations for mitigating them.

1.2 Contracts Assessed

Name	Contract	Live Code Match
`CosmicGuild`	`CosmicGuild.sol`	N/A
`CosmicGuildFundingContract`	`CosmicGuildFundingContract.sol`	N/A
`CosmicGuildStaking`	`CosmicGuildStaking.sol`	N/A

Note: The team did not get back to us with the deployed addresses for the other contracts, so users should take note when interacting with the contracts.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	1	1	-	-
Low	5	4	-	1
Informational	13	13	-	-
Total	**20**	**19**	**-**	**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 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 CosmicGuild

ID	Severity	Summary	Status
01	INFO	Ownable is not used in the contract and can be removed	✓ RESOLVED
02	INFO	`SUPPLY_CAP` can be made constant and value readability can be further improved	✓ RESOLVED
03	INFO	`uint256` instead of `uint` should be used for better interpretation of contract variables	✓ RESOLVED

1.3.2 CosmicGuildFundingContract

ID	Severity	Summary	Status
04	LOW	Governance risk: The contract owner can withdraw all the reward tokens and cause functions in the staking contract to revert	ACKNOWLEDGED
05	INFO	`resetAllowance` is unnecessarily public	✓ RESOLVED
06	INFO	`cg` and `CosmicGuildStaking` addresses can be made immutable	✓ RESOLVED
07	INFO	`cg` can be marked as IERC20 so it does not need to be cast it later	✓ RESOLVED

1.3.3 CosmicGuildStaking

ID	Severity	Summary	Status
08	HIGH	Deposits do not support tokens with a fee on transfer	✓ RESOLVED
09	MEDIUM	`emergencyWithdraw` and `deposit` do not adhere to checks-effects-interaction pattern and could be prone to reentrancy, allowing for reentrancy tokens to be completely stolen and the reward currency to be completely drained	✓ RESOLVED
10	LOW	`pendingCG` and `updatePool` will revert if `totalAllocPoint` is zero	✓ RESOLVED
11	LOW	No maximum safeguard for the emission rate per block	✓ RESOLVED
12	LOW	`startBlock` and `bonusEndBlock` can be set in the past	✓ RESOLVED
13	LOW	Deposits and withdrawals can revert when the funding contract runs out of token supply	✓ RESOLVED
14	INFO	`BONUS_MULTIPLIER` and `bonusEndBlock` functionality can be completely removed	✓ RESOLVED
15	INFO	Any address can be added as a pool in the staking contract	✓ RESOLVED
16	INFO	`cg` and `fundingContract` can be made immutable	✓ RESOLVED
17	INFO	`set`, `deposit`, `withdraw`, `emergencyWithdraw` can be made external	✓ RESOLVED
18	INFO	Rounding vulnerability to tokens with an extensive supply can cause large supply tokens to receive zero emissions	✓ RESOLVED
19	INFO	Lack of events for `add` and `set`	✓ RESOLVED
20	INFO	Typographical errors	✓ RESOLVED

2 Findings

2.1 CosmicGuild

`CosmicGuild` is an ERC20 token implementation that will be used in the project as a reward to be distributed to users that will be depositing tokens into the `CosmicGuildStaking` contract.

An ERC20 contract adapts the use of transferable financial instruments which can be bought and sold at a value determined by the market. This type of contract became standard in creating exchange methods in the blockchain space and thus used in most financially involved projects like yield farms.

The total supply of `CosmicGuild` is pre-minted via the constructor upon deployment at an address the deployer will input. All or a percentage of the tokens are expected to be transferred to `CosmicGuildFundingContract` as this contract will serve as temporary storage for tokens that will be distributed to depositors. The token also uses the Ownable library from OpenZeppelin that can restrict certain functions for the use of governance. However, privileged functions have yet to be added.

Overall, it is a straightforward token contract that premints its supply to an address and is utilized by its corresponding staking contract for distribution.

2.1.1 Token Overview

Address	TBD
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	500,000,000

2.1.2 Privileged Roles

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

- `transferOwnership`
- `renounceOwnership`

2.1.3 Issues & Recommendations

Issue #01	Ownable is not used in the contract and can be removed
Severity	INFORMATIONAL
Location	Line 6 `import "@openzeppelin/contracts/access/Ownable.sol";`
Description	Files imported in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length unnecessarily.
Recommendation	Consider removing the import as mentioned above to keep the contract short and simple.
Resolution	RESOLVED

Issue #02	SUPPLY_CAP can be made constant and value readability can be further improved
Severity	INFORMATIONAL
Location	Line 10 `uint SUPPLY_CAP = 500000000 ether;`
Description	Number values can use an underscore (_) as a thousand separator to increase readability. For example, `1_000 ether`, `10_000 ether`, `100_000 ether`, etc. As the SUPPLY_CAP does not change, it can also be made constant to reduce the gas cost of the contract as constants are encoded directly into the contract bytecode. As this variable is only used once, we do admit that gas optimization is hardly relevant. However, adhering to best practice still shows good code hygiene which is appreciated by third-party validators and reviewers.
Recommendation	Consider revising the values to the method as mentioned above.
Resolution	RESOLVED

Issue #03	**`uint256` instead of `uint` should be used for better interpretation of contract variables**
Severity	INFORMATIONAL
Description	We recommend remaining consistent throughout the project and only use `uint256`. Being consistent shows third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	RESOLVED

2.2 CosmicGuildFundingContract

`CosmicGuildFundingContract` is be a fund-storing contract for the project which stores `CosmicGuild` tokens. It acts like a simple bank dependent on the instructions to transfer out tokens to the staking contract for reward distribution.

Upon deployment, the `CosmicGuild` token address and the `CosmicGuildStaking` address will be initialized via the constructor to be able to connect the three contracts. An unlimited approval is granted upon deployment so that the staking contract can transfer tokens in by sending them from the funding contract to the staking contract. Transfer requests from the staking contract are done whenever it updates the pool information, and reward tokens need to be distributed to users as rewards.

Lastly, the governance can withdraw a specific amount of reward tokens in the contract, which can interfere with the staking contract functions. It is therefore crucial that the ownership of the contract is carefully safeguarded to prevent risk towards the `CosmicGuild` tokenomics.

2.2.1 Privileged Roles

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

- `withdraw`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #04	**Governance risk: The contract owner can withdraw all the reward tokens and cause functions in the staking contract to revert**
Severity	LOW SEVERITY
Description	The Funding Contract owner can, at any point in time, call `withdraw` to take back all `CosmicGuild` tokens from this contract. This could be a governance risk for users who assume those tokens are locked away. Additionally, since the staking contract is dependent on the remaining reward token balance in this funding contract, calling this function and inadvertently withdrawing all the tokens can interfere with the staking contract's functions.
Recommendation	Consider being forthright to your community on the purpose of using the function and when tokens will be withdrawn from the contract. This would at least establish trust and give advance notice to users participating in the project.
Resolution	ACKNOWLEDGED

Issue #05	`resetAllowance` is unnecessarily public
Severity	INFORMATIONAL
Description	The function to grant unlimited allowance to the `CosmicGuildStaking` contract as spender is open to the public. This function is already called via constructor upon deployment, and it is no longer necessary to call it again.
Recommendation	Consider simply granting infinite approval in the constructor through a standard `approve()` call.
Resolution	RESOLVED This function has been made `internal`.

Issue #06	`cg` and `CosmicGuildStaking` addresses 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 variables mentioned above explicitly immutable.
Resolution	RESOLVED

Issue #07	cg can be marked as IERC20 so it does not need to be cast it later
Severity	INFORMATIONAL
Description	Variables of type IERC20 can be marked as this if the contract imports the specific library.
Recommendation	Consider marking the variables as mentioned above of type IERC20.
Resolution	RESOLVED

2.3 CosmicGuildStaking

`CosmicGuildStaking` is a fork of PancakeSwap/SushiSwap's Masterchef. A notable feature of forking the latter is the removal of the `migrator` function, which can potentially be used to steal users' tokens.

The project does not implement a deposit fee feature and has settled with a much simpler and cleaner version of the contract. The governance can add/set token pools and their corresponding reward distribution rate and uses the conventional block number basis for reward calculation.

Reward tokens are sourced from the `CosmicGuildFundingContract`. The governance needs to therefore manually fund this funding contract with tokens beforehand. Rewards have a finite supply and are limited only to 500 million tokens based on the total supply of the `CosmicGuild` token (the total rewards will likely be much lower).

Users will be able to deposit, withdraw, and emergency withdraw their tokens. By depositing tokens, they will be able to earn rewards depending on how long their funds stay within the staking contract.

2.3.1 Privileged Roles

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

- `add`
- `set`
- `updateEmissionRate`
- `transferOwnership`
- `renounceOwnership`

2.3.2 Issues & Recommendations

Issue #08	Deposits do not support tokens with a fee on transfer
Severity	HIGH SEVERITY
Description	Within the `deposit` function, there is no logic that supports tokens with a fee on transfer. Therefore, during a deposit, the masterchef will receive fewer tokens than the user will get credited. This could be exploited, where a malicious user can drain the whole pool in case a token with a fee on transfer is added, which results in absurd reward minting.
Recommendation	Consider adding logic for tokens with a fee on transfer. This logic should check the balance before depositing and recheck it afterward. The increase in balance between these two checks represents the deposited amount. Reentrancy guards should always accompany such a mechanism.
Resolution	RESOLVED The client has added the recommended before-after logic.

Issue #09	**`emergencyWithdraw` and `deposit` do not adhere to checks-effects-interaction pattern and could be prone to reentrancy, allowing for reentrancy tokens to be completely stolen and the reward currency to be completely drained**
Severity	MEDIUM SEVERITY
Description	The `emergencyWithdraw` function updates the `user.amount` only after the tokens have been transferred to the user. In case an exploiter is able to execute code during this outbound transfer in what is called a reentrancy attack, the exploiter would be able to call `emergencyWithdraw` again to withdraw their balance twice. Within `deposit`, the `rewardDebt` is updated only after the funds are transferred in: this allows an exploiter to call `deposit` multiple times if a reentrancy vector is permitted to harvest their pending harvest multiple times. Adhering to the checks-effects-interactions pattern can further minimize the attack surface following an external contract call: https://docs.soliditylang.org/en/latest/security-considerations.html This issue is marked as medium severity as we do not expect Cosmic Guild to add such tokens. In case Cosmic Guild plans to add special tokens, we highly recommend that this issue is properly addressed.
Recommendation	Consider rearranging the affected lines of code to follow the checks-effects-interactions pattern. Consider adding reentrancy guards to at least `deposit`, `withdraw` and `emergencyWithdraw`.
Resolution	RESOLVED The client has rewritten `emergencyWithdraw` to adhere to checks-effects-interactions and has added reentrancy guards to all three functions to prevent potential exploitation of the `deposit` function.

Issue #10	**pendingCG and updatePool will revert if totalAllocPoint is zero**
Severity	LOW SEVERITY
Description	Within the pendingCG and updatePool functions, at some point a division is made by the totalAllocPoint variable. If all pools have their rewards set to zero, this variable will also be zero. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards starting from 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.timestamp > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {` Within updatePool, the totalAllocPoint check can be added to the lpSupply if-statement (it is important that the last update block is still updated).
Resolution	RESOLVED The client has implemented the recommended codeblock for pendingCG and the recommended check for totalAllocPoint in updatePool.

Issue #11	No maximum safeguard for the emission rate per block
Severity	LOW SEVERITY
Description	The function to update rewards has no safeguard on its maximum value. Projects sometimes accidentally update their emission rate to a severely high number either by accident or malicious intent. By having a maximum value, the code itself enforces the reward rate to always be within a reasonable range, preventing mistakes that cannot be reverted once made. This might boost investor confidence as they know that the governance cannot suddenly set the emission rate to a very high value.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_cgPerBlock <= MAX_EMISSION_RATE,"Too high");` This check should be done within the constructor as well.
Resolution	RESOLVED A safeguard has been added in both the constructor and the emission adjustment function: it is set to 500 tokens per block.

Issue #12	`startBlock` and `bonusEndBlock` can be set in the past
Severity	LOW SEVERITY
Description	`startBlock` and `bonusEndBlock` when input in the constructor have no restrictions and can be set to any block which could complicate reward calculation.
Recommendation	Consider adding requirements to the constructor inputs where `startBlock` and `bonusEndBlock` should be greater than the current `block.number` and where `bonusEndBlock` should also be greater than the `startBlock`. It should be noted that `bonusEndBlock` should be removed completely in our opinion.
Resolution	RESOLVED `bonusEndBlock` has been removed while `startBlock` is now validated.

Issue #13	**Deposits and withdrawals can revert when the funding contract runs out of token supply**
Severity	LOW SEVERITY
Description	Currently, the token supply is 500 million, and there are no checks implemented where the limit is reached before the transfer. This could mean that for example, if the current supply is 499,999,999 tokens and there is a request to transfer two tokens, this request will not pass as there is no balance in the funding contract. More realistically, the staking contract sources tokens from the funding contract, which might run out of funds much more quickly than this theoretical limit.

Recommendation	Consider reducing the reward to transfer according to how much is remaining in the funding contract when the staking contract calls on the `updatePool` function.
	
	`uint256 remaining = cg.balanceOf(address(fundingContract));`
	`if (cgReward > remaining) {`
	`  cgReward = remaining;`
	`}`

Resolution	RESOLVED The recommended codeblock has been implemented.

Issue #14	**BONUS_MULTIPLIER and bonusEndBlock functionality can be completely removed**
Severity	INFORMATIONAL
Description	The constant variable `BONUS_MULTIPLIER` alongside its related `bonusEndBlock` variable do not contain any extra information since the bonus is fixed at 1 (no bonus). These variables are therefore redundant and might mislead third-party reviewers into thinking that there is such a thing as bonus multipliers within the contract logic.
Recommendation	Consider removing the `BONUS_MULTIPLIER` and `bonusEndBlock`. The `getMultiplier` function can then be significantly simplified: `return _to.sub(_from);`
Resolution	RESOLVED The complex logic has been removed in favor of the recommended simplified `getMultiplier` function.

Issue #15	**Any address can be added as a pool in the staking contract**
Severity	INFORMATIONAL
Description	When calling the add function, the owner may inadvertently input a non-ERC20 contract into the pool details and push an incorrect pool into the masterchef. This would result in unnecessary reward calculation and re-working of pool addition.
Recommendation	Consider adding the following line at the start of the add function to make sure that the contract is a valid ERC20 token contract: `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED The recommendation has been implemented.

Issue #16	**cg and `fundingContract` can be made immutable**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated 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 variables as mentioned above explicitly immutable. It should be noted that to make `fundingContract` immutable, it should first be assigned to a local variable in the constructor, then the local variable should be used for the ownership transfer. This is to avoid the compiler from complaining that you are reading immutable variables in the constructor.
Resolution	RESOLVED

Issue #17	**`set`, `deposit`, `withdraw`, `emergencyWithdraw` can be made external**
Severity	INFORMATIONAL
Description	Functions not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to lower gas usage in certain cases.
Recommendation	Consider marking the functions mentioned above as external.
Resolution	RESOLVED

Issue #18	Rounding vulnerability to tokens with an extensive supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL

Description	Within `updatePool`, `deposit`, `withdraw`, and the `pendingRewards` function, `accCGPerShare` is based upon the `lpSupply` variable.
	
	`pool.accCGPerShare = pool.accCGPerShare.add(`
	`  cgReward.mul(1e12).div(lpSupply)`
	`);`
	
	However, if this `lpSupply` becomes a severely large value, this will cause precision errors 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. It should be noted that even a precision of 1e18 has been considered small when meme tokens were added to the masterchefs of our client. If the client thinks it is realistic that such tokens will be added, we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	RESOLVED

Issue #19	Lack of events for add and set
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

Issue #20	Typographical errors
Severity	INFORMATIONAL

Description	Line 9
	`import "./CosmicGuild.sol";`
	
	The `CosmicGuild` import can be removed as only standard IERC20 functionality is used. The `cg` variable can therefore be changed to an immutable IERC20 without any issue. This would reduce the code visible in the explorer and make it easier for third parties to validate said code.
	
	Line 47
	`// Bonus muliplier for early cg makers.`
	
	Multiplier has been misspelled.
	
	Line 54
	`// Total allocation poitns. Must be the sum of all allocation points in all pools.`
	
	Points have been misspelled.
	
	Line 170
	`// Update reward vairables for all pools. Be careful of gas spending!`
	
	Variables have been misspelled.
	
	Line 214
	`address(msg.sender),`
	
	Throughout the contract, the client casts `msg.sender` to `address(msg.sender)`. As `msg.sender` is already of the address type, this is unnecessary. Consider replacing all occurrences.

	Finally, although not strictly necessary with the current CosmicGuild token, the client should consider either using `SafeERC20` or handling the return variables of the token transfers within `safeCGTransfer`.
Recommendation	Consider fixing the issues as mentioned above.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY