MoonPool

SunnyDefi Project

This is a free audit report

Published on 7th April, 2021

SunnyDefi Security Analysis

By MoonPool

Overview Abstract

In this report, we consider the security of the SunnyDefi project. Our main task is to find and describe security issues in the smart contracts of the platform to the team.

Limitations and use of the report

Broadly speaking, the assessments can not uncover all vulnerabilities of the given smart contracts, thus, it is not guaranteed that the system is secured even if no vulnerabilities are found. The focus of the assessments was limited to only 3 given smart contracts, other contracts were excluded (including external libraries or third party codes).

Note: The report is not investment advice.

The team can publish this report to the community.

Summary

We have found 1 medium severity issue and 8 low severity issues.

The source codes were mainly influenced from well-known projects on Ethereum and Binance Smart Chain.

Recommendations

We recommend the team to fix all issues, as well as tests coverage to ensure the security of the contracts.

Assessment Overview

Scope of the audit

Source codes for the audit was initially taken from the commit hash *a5d580c735e7e795d699897d6688d8600009b4de* and deployed contracts on Binance Smart Chain with the following **3** contracts:

1. ***SunnyToken.sol***
2. ***MasterChef.sol***
3. ***Timelock.sol***

Other source codes are out of scope for this report.

System Overview

The system overview is based on the provided documentation that is available on their website here .

1. Sunny Token :

- A standard token that allows the owner to mint more tokens to a given address with a given amount. A 500 tokens will be pre-minted after the contract has been deployed.

- The owner of the contract will be set to the MasterChef, which allows only MasterChef to mint more token.

2. MasterChef

- Allow stakers to deposit the supported tokens (single tokens or LP tokens) to the contract and earn Sunny token as the reward.
  Each supported token/pool will have different configurations, include:
- **allocPoint** : how many allocation points that are assigned to this token/pool. The Sunny reward will be distributed for stakers of this token/pool based on the **allocPoint** .
- **depositFeeBP** : How many percent (in basis points) of deposited tokens will go to the fee address. 1 is 0.01% and 10,000 is 100%. The **depositFeeBP** is at most **10%**

4. Timelock

A standard timelock contract that allows the owner to submit any transactions which are only executed after the **delay** time. The minimum delay time is set as **6 hours**. The minimum delay is also 6 hours.

Findings

We have found 1 medium severity issue and 8 low severity issues.

SunnyToken

[Low] Potentially high gas consumption for transfer, transferFrom, mint, etc

This code is influenced by Compound token contract code to allow taking a snapshot on each action (transfer, transferFrom, mint, etc), that will be used to calculate voting power for the governance.
If the team does not want to use the token for their Governance, we recommend removing the snapshot mechanism to reduce gas consumption.

[Low] Code Convention

1. Function order is incorrect. Mint function should be declared below state variable declaration, function order should be: external -> public -> internal.
2. Error message for require is too long: Some requires are using too long error messages, consider to make them shorter.
3. Line 978: Should avoid to use inline assembly.

MasterChef

[Medium] updateEmissionRate logic is different from comment in the code.

In the comment, it states that the emission rate is reduced 3% every 14,400 blocks, which is around 12 hours.
However, in the logic of the **updateEmissionRate** function, the **EMISSION_REDUCTION_PERIOD_BLOCKS** is set to be 7,200 blocks, which is around 6 hours, and the emission rate is reduced **EMISSION_REDUCTION_RATE_PER_PERIOD = 5%** every 7,200 blocks.
As the result, either the comment or the implementation is incorrect.

[Low] Redundant check for _referrer address to be address(_referrer)

In line 1596 of the **deposit** function with referral, it requires **_referrer == address(_referrer)** which is a redundant check. It also happens in line 1719 of the function **setReferral**.

[Low] Redundant casting msg.sender to address type

In the line 1645 of the **withdraw** function, it casts msg.sender to an address type, which is redundant since the msg.sender is already of type address.

[Low] Compress PoolInfo struct data to save gas

According to the requirements:

- **allocPoint** is not greater than 10,000, which is fit in 14 bits.
- **lastRewardBlock** : is the block number, which should be easily fit in 32 bits.
- **accSunnyPerShare** : is the accumulated Sunny token per share, times 10^12. It can be fit in a uint128.

Consider to use uint16 for **allocPoint** and uint32 for **lastRewardBlock** , uint128 for **accSunnyPerShare** so that (**allocPoint, lastRewardBlock, accSunnyPerShare, depositBP)** should be fit in a single uint256, thus, reducing storage save and load.

[Low] Compress UseInfo struct data to save gas

UserInfo has only 2 attributes of uint256: **amount** and **rewardDebt.** Both values should be fit in uint128, thus, can compress the **amount** and **rewardDebt** to a single uint256.

[Low] Reset user's data before transferring LP tokens instead of using non-reentrancy guard

In the **emergencyWithdraw** function, should reset user's data before calling transfer LP tokens to users, which will help to prevent reentrancy attack, thus, can remove *nonReentrant* modifier to save gas.

[Low] Code Convention

1. Error message for require is too long: Some requires are using too long error messages, consider to make them shorter.

2. Lack of comments in the code: Should add more comments to the code, especially for each function to describe its functionality.

3. Line length is too long: Line length shouldn't be too long, length <= 100 should be good for reading.

4. Function order is incorrect: Functions should be in the order: external -> public -> internal -> private and write -> read functions.

Timelock

This is a standard and common Timelock contract

Testing

The team has not provided any tests for given contracts. We recommend adding full tests coverage to ensure the logic works as expected and the security of the smart contracts.