For Red Apple
29 Nov 2022

MAXLOOP

Smart Contract Security Assessment

Final Report

Maxloop.org Maxloop.org@gmail.com Telegram/MaxLoop

Disclaimer

Maxloop Blockchain Security ("Maxloop") 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 Maxloop.

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 Maxloop is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Maxloop 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 Red apple on the Binance Smart Chain (BSC). Maxloop 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	Red Apple
URL	http://www.redapple.com.co/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
Red apple	**0x4A2a4Be19456132b0bc3dc67D678F55799448258**	MATCH
Red apple.sol	**bc33e475e3c1cb1ba0851246fc15ec2e90fc03e6cbef22c7719289deabab6daa**	MATCH

Audit Summary

Delivery Date	Feb. 29th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Nov. 27, 2022 - Nov. 29, 2022

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	3	1	-	2
Informational	1	-	-	1
Total	**4**	**1**	**-**	**3**

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 Red appleToken

ID	Severity	Summary	Status
01	LOW	Mint function can be used to pre-mint large amounts of tokens before Ownership is transferred to the **Masterchef**	RESOLVED

1.3.2 Red appleMasterChef

ID	Severity	Summary	Status
01	LOW	Inconsistency between deposit fee cap in add and set	ACKNOWLEDGED
02	LOW	PendingMondo will show inaccurate pending harvests on the dapp frontend If the pending rewards causes totalSupply to be exceed **MAXSUPPLYCAP**	ACKNOWLEDGED
03	INFO	Total token supply might not be minted due to try and catch pattern	ACKNOWLEDGED

2 Findings

2.1 Red appleToken

The red apple token is a simple BEP-20 token which will be used as the main reward token for the Masterchef. The contract allows for red apple tokens to be minted when the mint function is called by the contract Owner, who at the time of deployment would be the deployer. Ownership is generally transferred to the Masterchef via the transferOwnership function for emission rewards to be minted and distributed to users staking in the Masterchef. The token has a max supply cap of 96,988,560,707(minting).

2.1.1 Token Overview

Address	**0x4A2a4Be19456132b0bc3dc67D678F55799448258**
Token Supply	96,988,560,707 (Minting)
Decimal Places	18
Transfer Max Size	1%
Liquidity	6%
Reflections	2%
Wallet Max Size	2%

2.1.2 Privileges

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

- mint
- renounceOwnership
- transferOwnership

2.1.3 Issues & Recommendations

Issue #01	**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 premints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this mint function is to be 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 1 tokens were pre-minted and ownership has been transferred to the Masterchef.

2.2.1 Privileges

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

- add
- set
- updateEmissionRate
- updateStartTimestamp
- transferOwnership
- renounceOwnership

The following functions can be called by the DevAddr of the Masterchef:

- setDevAddress

The following functions can be called by the FeeAddr of the Masterchef:

- setFeeAddress

2.2.2 Issues & Recommendations

Issue #02	Inconsistency between deposit fee cap in add and set
Severity	LOW SEVERITY

Description	For deposit fees, while add has a max cap of 8%, set has a max cap of 4%. add: Line 1136 require(_depositFeeBP <= 800, "add: invalid deposit fee basis points”); set: Line 1159 require(_depositFeeBP <= 400, "set: invalid deposit fee basis points"); This behavior is inconsistent, and could allow the owner to add a pool with 8% fee, even if 4% is the expected maximum cap.
Recommendation	The red apple team should clarify what their maximum cap on the deposit fee is, and ensure that checks in both add and set use the same value. It is encouraged to use the lower value as the cap.
Resolution	ACKNOWLEDGED

Issue #03	**pendingred apple will show inaccurate pending harvests on the dapp frontend if the pending rewards causes totalSupply to be exceed MAXSUPPLYCAP**
Severity	LOW SEVERITY

Location	Similarly to updatePool, pendingred apple does not check if the pending rewards will cause the total supply to exceed the MAXSUPPLYCAP. This can cause inaccurate pending harvests to be shown towards the end of token emissions.

Description	Consider factoring in the MAXSUPPLYCAP, and set the pending reward to be the difference between MAXSUPPLYCAP and totalSupply if the pending reward causes totalSupply to exceed MAXSUPPLYCAP.
	
	`uint256 red appleReward = multiplier.mul(red applePerBlock).mul(pool.allocPoint).div(total AllocPoint);`
	
	`if (red apple.totalSupply().add(red appleReward) > red apple.maxSupply()) { red appleReward = red apple.maxSupply() .sub(red apple.totalSupply()); }`
	
	`accred applePerShare = accred applePerShare.add(red appleReward.mul(1e18).div(pool.lpSupply ));`

Recommendation	pendingred apple will show inaccurate pending harvests on the dapp frontend if the pending rewards causes totalSupply to be exceed MAXSUPPLYCAP.
Resolution	ACKNOWLEDGED

Issue #04	**Total token supply might not be minted due to try and catch pattern**

Severity	INFORMATIONAL

Description	As there is a MAXCAPSUPPLY for the red apple token, minting the reward and causing the maximum cap to exceed would result in a revert.
	red appleToken::Line 814: require(_totalSupply.add(amount) <= MAXCAPSUPPLY, "Max supply reached");
	To prevent this, the following try and catch pattern is done in updatePool.
	Line 1209~
	try red apple.mint(devaddr, red appleReward.div(10)) {
	} catch (bytes memory reason) { red appleReward = 0;
	emit red appleMintError(reason);
	}
	try red apple.mint(address(this), red appleReward) {
	} catch (bytes memory reason) { red appleReward = 0;
	emit red appleMintError(reason); }
	In the case where totalSupply + amount does exceed MAXCAPSUPPLY, the mint will not be done. This means that the token supply could be capped at an amount slightly lower than MAXCAPSUPPLY.

Recommendation	Consider minting the difference between MAXCAPSUPPLY and totalSupply, if any.
	uint256 red appleReward = multiplier.mul(red applePerBlock).mul(pool.allocPoint).div(total AllocPoint);
	uint256 devReward = red appleReward.div(10); uint256 totalRewards = red apple.totalSupply().add(devReward).add(red appleReward);

```
if (totalRewards  <= red apple.maxSupply()) {
    //  mint dev reward as normal as not at maxSupply
    red apple.mint(devaddr, devReward);
} else {
    // update red appleReward to difference
    red appleReward= red apple.maxSupply() - red apple.totalSupply();
}
if (red appleReward != 0) {
    // only mint to MC and calculate and update accred applePerShare if red
appleReward is non 0     red apple.mint(address(this), red appleReward);
pool.accred applePerShare = pool.accred applePerShare.add(red
appleReward.mul(1e18).div(pool.lpS upply)); }
pool.lastRewardBlock = block.number;
```

Resolution ACKNOWLEDGED


MAXLOOP