Magpie

Security Audit Report

July 15, 2024

Contents

1 Introduction **3**

1.1 About Magpie . . . 3

1.2 Audit Scope . . . 3

1.3 Changelog . . . 3

2 Overall Assessment **4**

3 Vulnerability Summary **5**

3.1 Overview . . . 5

3.2 Security Level Reference . . . 6

3.3 Vulnerability Details . . . 7

4 Appendix **13**

4.1 About AstraSec . . . 13

4.2 Disclaimer . . . 13

4.3 Contact . . . 13

1 Introduction

1.1 About Magpie

The Magpie DAO is a suite of protocols that transcends conventional platforms with the goal of enhancing participation and user opportunities across DeFi. Integrated with an array of structures and solutions, Magpie adapts itself to different frameworks with the goal of achieving continuous growth while supporting the core functionalities of the decentralized finance ecosystem. As a Mega DAO composed by various SubDAOs, Magpie focuses on blackholing governance tokens, offering liquid restaking services for Ethereum and liquid staking for Bitcoin.

1.2 Audit Scope

	LINK	Base Commit	Final Commit
1	https://github.com/magpiexyz/magpie_contracts/tree/Launchpad	7aacd21	80463d7
2	https://github.com/magpiexyz/magpie_contracts/pull/168	fefcf60	be88d54
3	https://github.com/magpiexyz/magpie_contracts/pull/187	a337217	dc0d8ab

1.3 Changelog

Version	Date
First Audit	December 11, 2023
Second Audit	February 7, 2024
Third Audit	July 15, 2024

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Magpie` project. Throughout this audit, we identified several issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	2	1	-	1
Low	3	1	-	2
Informational	1	-	-	1
Undetermined	-	-	-	-

3 | Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~ Revised Update of maxToDistribute in Launchpad::configLaunchpad()

M-2 Potential Risks Associated with Centralization

L-1 Revised Update of startTime in Launchpad::configLaunchpad()

~~L-2~~ Improved Validation Checks in _checkValidCapAndUpdate()
~~L-3~~ Revisited Rewards Distribution Logic in ARBRewarder::setPool()

~~I-1~~ Improved Gas Efficiency in ARBRewarder::_calculateAndSendARB()

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[M-1] Revised Update of maxToDistribute in Launchpad::configLaunchpad()

Target	Category	IMPACT	LIKELIHOOD	STATUS
Launchpad.sol	Business Logics	Medium	Medium	Addressed

In the `Launchpad` contract, the project tokens are funded by the owner via the `configLaunchpad()` routine. The owner can adjust the maximum amount of project tokens to distribute before the start of the launchpad. Specially, if the owner increases the maximum amount (line 379), it needs to fund the increased amount of project tokens to the contract (line 380).

However, we notice it's possible that the owner may reduce the maximum amount, and there is a lack of refund for the decreased amount of project tokens back to the owner. Without the refund, the decreased amount of project tokens can't be withdrawn normally via the `withdrawUnsoldTokens()` routine, unless via emergency withdraw.

Launchpad::configLaunchpad()

```
374  maxRaiseAmount = _maxToRaise;
375  LOW_FDV_VESTING_PART = _lowFDVVestingPart;
376  HIGH_FDV_VESTING_PART = _highFDVVestingPart;
377  min_sale_token_amount = _minSaleTokenAmount;

379  if (_maxToDistribute > max_launch_tokens_to_distribute) {
380      IERC20(projectToken).safeTransferFrom(msg.sender,address(this),
           _maxToDistribute - max_launch_tokens_to_distribute);
381  }
382  max_launch_tokens_to_distribute = _maxToDistribute;
```

Remediation Promptly refund the decreased amount of project tokens back to the owner if it decreases the maximum amount of project tokens to distribute.

[M-2] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Launchpad.sol	Security	Medium	Medium	Acknowledged

In the `Magpie Launchpad` protocol, the existence of a privileged owner account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the owner.

Launchpad::emergencyWithdrawFunds()

```
415 function emergencyWithdrawFunds(address token, uint256 amount) external
        whenPaused onlyOwner {
416     IERC20(token).safeTransfer(owner(), amount);
418     emit EmergencyWithdraw(token, amount);
419 }
```

Remediation To mitigate the identified issue, it is recommended to designate a multi-sig account to undertake the role of the privileged `owner` account. Moreover, it is advisable to implement timelocks to govern all modifications to privileged operations.

Response By Team This issue has been confirmed by the team.

[L-1] Revised Update of startTime in Launchpad::configLaunchpad()

Target	Category	IMPACT	LIKELIHOOD	STATUS
Launchpad.sol	Coding Practice	Medium	Low	Acknowledged

The `Launchpad` contract allows the owner to configure critical parameters for the launchpad, including the start time, project token, sale token, etc. Meanwhile, the owner can add new sale phases which shall be started later than the launchpad.

In particular, the owner can re-configure the launchpad before it is started as long as the new start time is sometime in future (line 357). However, we notice there is a lack of validation for the new start time to ensure it will be started before the first sale phase. As a result, if the launchpad is started after some sale phases, user can't participate in these overdue sale phases.

Launchpad::configLaunchpad()

```
357 if (_startTime <= _currentBlockTimestamp()) revert InvalidTime();
358 if (_projectToken == address(0)) revert ZeroAddress();
359 if (_saleToken == address(0)) revert ZeroAddress();
360 if (_vestingContract == address(0)) revert ZeroAddress();
361 if (_lowFDVVestingPart >= DENOMINATOR) revert InvalidFDVPart();
362 if (_highFDVVestingPart >= DENOMINATOR) revert InvalidFDVPart();

364 uint8 projectTokenTokenDecimals = IERC20Metadata(_projectToken).decimals();
365 uint8 saleTokenDecimals = IERC20Metadata(_saleToken).decimals();

367 if (saleTokenDecimals > projectTokenTokenDecimals) revert
        TokenDecimalExceedsLimit();
```

```
369 projectToken = _projectToken;
370 saleToken = _saleToken;
371 vestingContract = ILaunchpadVesting(_vestingContract);
372 startTime = _startTime;
```

Remediation Add a check for the new start time of the launchpad and ensure it will be started prior to the first sale phase (if any).

Response By Team The issue has been confirmed by the team and they will examine the parameters.

[L-2] Improved Validation Checks in _checkValidCapAndUpdate()

Target	Category	IMPACT	LIKELIHOOD	STATUS
Launchpad.sol	Coding Practices	Low	Low	Addressed

The `Launchpad` contract provides an external `buy()` function for users to purchase a `PROJECT Token` allocation for the sale with a value of "amount" `saleToken`. This function requires that the provided amount of saleToken must not be less than `min_sale_token_amount`, but there is no corresponding check for the amount of `PROJECT Token` that the user can get in return.

To elaborate, we show below the implementation of the `_checkValidCapAndUpdate()` function. Our analysis shows that `_checkValidCapAndUpdate()` can be strengthened by further ensuring `require(amountOfTokensToBeAllocated > 0, 'Invalid allocation')` so that the buyer will not have potential asset loss.

Launchpad::_checkValidCapAndUpdate()

```
441 function _checkValidCapAndUpdate(uint256 _saleTokenAmount, uint256 _phaseNumber)
        internal {
442     PhaseInfo storage phaseInfo = phaseInfos[_phaseNumber - 1];

444     totalRaised += _saleTokenAmount;

446     if (totalRaised > maxRaiseAmount) revert RaisedMaxAmount();

448     uint256 amountOfTokensToBeAllocated = _tokenAllocBySale(_saleTokenAmount,
            phaseInfo);

450     totalAllocated += amountOfTokensToBeAllocated;
451     phaseInfo.allocatedAmount += amountOfTokensToBeAllocated;

453     if (
454         totalAllocated > max_launch_tokens_to_distribute
```

```
455             totalAllocated > phaseInfo.saleCap
456         ) revert NotEnoughToken();
457     }
```

Remediation Add proper check to ensure the user can receive a reasonable amount of project token.

[L-3] Revisited Rewards Distribution Logic in ARBRewarder::setPool()

Target	Category	IMPACT	LIKELIHOOD	STATUS
ARBRewarder.sol	Business Logic	Medium	Medium	Addressed

The `ARBRewarder` contract provides a public `setPool()` function for the privileged `owner` account to modify the key parameters of a staking pool. When examining its implementation logic, we notice that the distribution of pending rewards is incorrect under certain conditions.

To elaborate, we show the related code snippet below. Specifically, when the pool status remains active before and after parameter modification, but the `masterChef` address changes, the pending `ARB` rewards should be sent to the `rewarder` address obtained from the old `masterChef`, rather than being stored in the current contract and sent to the `rewarder` address obtained from the newly set `masterChef` during the next harvest operation.

ARBRewarder::setPool()

```
212     function setPool(address _stakingToken, address _masterChef, bool _isActive,
            uint256 _endTimestamp) external onlyOwner {

214         if(_masterChef == address(0))
215             revert ZeroAddress();
216         if (_endTimestamp < block.timestamp)
217             revert InvalidEndtime();

219         PoolInfo storage pool = tokenToPoolInfo[_stakingToken];

221         if (pool.isActive && !_isActive) {// if setting an active pool to
                inactive, queue the pending arb rewards to rewarder
222             address rewarder = IMasterMagpie(pool.masterChef).getRewarder(
                    _stakingToken);
223             _calculateAndSendARB(_stakingToken, rewarder);
224         }
225         if (!pool.isActive && _isActive) {// if setting an inactive pool as
                active, just set the current timestamp as lastRewardTimestamp
226             pool.lastRewardTimestamp = block.timestamp;
227         }
```

```
228         pool.masterChef = _masterChef;
229         pool.isActive = _isActive;
230         pool.endTimestamp = _endTimestamp;

232         emit SetPool(_stakingToken, _masterChef, _isActive, _endTimestamp);
233     }
```

Remediation For the scenario described above, timely send the pending `ARB` rewards to the `rewarder` address obtained from the old `masterChef` address.

[I-1] Improved Gas Efficiency in ARBRewarder::_calculateAndSendARB()

Target	Category	IMPACT	LIKELIHOOD	STATUS
ARBRewarder.sol	Coding Practices	N/A	N/A	Addressed

The helper function `_calculateAndSendARB()` of the `ARBRewarder` contract is designed to calculate the pending `ARB` rewards and send the new rewards to be distributed to the `rewarder` contract. When examining its code implementation, we notice that the current gas usage could be further optimized. Specifically, by changing `tokenToPoolInfo[_stakingToken].lastRewardTimestamp = block.timestamp` to `pool.lastRewardTimestamp = block.timestamp` (line 135), the number of direct accesses to the state variable can be reduced, thereby saving gas.

ARBRewarder::_calculateAndSendARB()

```
115     function _calculateAndSendARB(address _stakingToken, address _rewarder)
            internal {

117         if(_rewarder == address(0))
118             return ;
119         PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
120         if(!pool.isActive  pool.ARBPerSec == 0 )
121             return ;

123         uint256 multiplier = block.timestamp - pool.lastRewardTimestamp;
124         if (block.timestamp >= pool.endTimestamp){
125             pool.isActive = false;
126             multiplier = pool.endTimestamp - pool.lastRewardTimestamp;
127         }
128         uint256 rewardAmount = (multiplier * pool.ARBPerSec);
129         rewardAmount = Math.min(rewardAmount, ARB.balanceOf(address(this)));

131         ARB.approve(_rewarder, rewardAmount);
132         IBaseRewardPool(_rewarder).queueNewRewards(rewardAmount, address(ARB));
```

```
134             emit ARBRewadsSent(_stakingToken, _rewarder, rewardAmount, pool.
                    lastRewardTimestamp, pool.ARBPerSec);
135             tokenToPoolInfo[_stakingToken].lastRewardTimestamp = block.timestamp;
136         }
```

Remediation Reduce the number of direct accesses to the state variable by using `pool.lastRewardTimestamp = block.timestamp` instead of `tokenToPoolInfo[_stakingToken].lastRewardTimestamp = block.timestamp` (line 135).

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI