Security Audit Report for Earnmos Vault and StratDiffusion Contracts

Date: May 03, 2022

Version: 1.0

Contact: contact@blocksecteam.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.05.03 20:34:51 +08'00'

Contents

Report Manifest

Item	Description
Client	Earnmos
Target	Earnmos Vault and StratDiffusion Contracts

Version History

Version	Date	Description
1.0	May 03, 2022	First version

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

For the current contracts covered in this audit, the asset deposited in the vault is the lp token of the `Diffusion` DEX [1]. The vault will deposit the lp token into the `Chef` contract (out of the scope for this audit) and earn the `DIFF` token. The `DIFF` token will be exchanged to the underlying tokens. The underlying tokens will be converted into the `Diffusion` lp token by adding liquidity into the DEX pool. The `Diffusion` lp token will again be deposited into the vault, enjoying the compound earnings. This architecture can support different vaults and strategies. For the current version of this audit, it only supports the strategy of `Diffusion` DEX.

The repository that has been audited is on github [2]. The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. Our audit report is responsible for the only initial version (`Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Earnmos Vault and StratDiffusion Contracts	`Version 1`	`a61800e7b2370ba0fa361a6bad56b3d6848a89ab`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the

[1] https://app.diffusion.fi

[2] https://github.com/earnmos/earnmos

computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Access control
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item

- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.
- **Fixed** The issue has been confirmed and fixed by the client.

[3] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[4] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find **one** potential issue. We have **one** recommendation.

- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Recommendations: 1

ID	Severity	Description	Category	Status
1	Low	Possible price manipulation attack	DeFi Security	Confirmed
2	-	Ensure the security of the private key of the owner	Recommendation	Confirmed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Possible price manipulation attack

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The function `harvest` of `StratDiffusion` swaps the `DIFF` token for the underlying tokens (line 155 and 164). It's susceptible to the price manipulation attack.

```
140  function harvest() public override whenNotPaused {
141    if (balanceOfPool() > 0) {
142      // claim diff
143      miniChef.harvest(pid, address(this));
144
145      uint256 diffBal = diff.balanceOf(address(this));
146      if (diffBal > 0) {
147        // charge fees
148        uint256 harvestFee = diffBal.mul(harvestFeeRate).div(
149        FEE_PRECISION
150        );
151        diff.safeTransfer(feeRecipient, harvestFee);
152
153        // swap diff
154        uint256 diffBalHalf = diffBal.sub(harvestFee).div(2);
155        if (lpToken0 != diff) {
156          uniRouter.swapExactTokensForTokens(
157          diffBalHalf,
158          0,
159          diffToLpToken0,
160          address(this),
161          now
162          );
```

```
163         }
164         if (lpToken1 != diff) {
165           uniRouter.swapExactTokensForTokens(
166           diffBalHalf,
167           0,
168           diffToLpToken1,
169           address(this),
170           now
171           );
172         }
173
174         // Adds liquidity and gets more want tokens.
175         uniRouter.addLiquidity(
176           address(lpToken0),
177           address(lpToken1),
178           lpToken0.balanceOf(address(this)),
179           lpToken1.balanceOf(address(this)),
180           1,
181           1,
182           address(this),
183           now
184         );
185
186         // reinvest
187         deposit();
188
189         emit Harvested(diffBal, harvestFee);
190       }
191     }
192   }
```

Listing 2.1: StratDiffusion.sol

Specifically, the `DIFF` token will be transferred into the strategy contract, and then swapped to the underlying tokens in the DEX pools. However, the price of the `DIFF` could be manipulated in the DEX pool so that it's price is very low and the underlying token exchanged will be less than expected. Besides, there is no slippage protection when swapping the `DIFF` for the underlying token. A possible attack is described in the following.

- The attacker borrows a large number of `DIFF` tokens using the flash loan.
- Then the attacker swaps the `DIFF` token to the underlying token in DEX pools.
- Then the attacker invokes the `harvest()` function. The function will get the `DIFF` token from the `Chef` contract and then swap the `DIFF` token to the underlying token. This will further lower the price of `DIFF` token.
- The attacker then makes a swap of the underlying token to `DIFF` token and then return back the `DIFF` token borrowed from flash loan.

The `deposit()` function in the vault contract can also invoke the `harvest` function. Note that, this attack only becomes profitable when the number of the `DIFF` token getting from the `Chef` contract is large. If the `harvest` function is invoked in a high frequency (e.g., through a bot from the project), the accumulated `DIFF` token in the `Chef` contact will be small. Due to this reason, we treat this as a low severity issue.

Impact This can make the exchanged underlying token less than expected.

Suggestion Ensure that only the EOA account can invoke the `deposit` function in the vault contract and the `harvest` function in the strategy. A slippage protection could be leveraged in the function. Besides, the project can monitor the accumulated `DIFF` token in the `Chef` contract and invoke the `harvest` function when the accumulated `DIFF` token exceeds a threshold.

Feedback from the Project We didn't put any limitation for the harvest() function because we want to decentralize this process. We allow anybody in the market to provide bots to call the Harvest() function. This will make the process more reliable and more profitable strategy compared to using some limitation to make it a permission function. Second, by doing so, we believe that the harvest will be executed very frequently and it will swap only a very small amount of the reward token for each time. It's not very economic to attack with a very small swap amount Third, by doing the above, we believe that all the bots will increase the gas to win the chance to call the Harvest() function. This will increase the cost of the attackers which will make their attack not profitable.

2.2 Additional Recommendation

2.2.1 Ensure the security of the private key of the owner

Status Confirmed

Description Since the contract owner can get all the `want` tokens in the vault, it's critical to ensure the security of the owner's private key. For instance, the multi-sig wallet can be used for the owner, and the hardware-based private key protection schema (e.g., TEE based solution) can be leveraged.

Impact NA.

Suggestion Ensure the security of the private key of the contract owner.

Feedback from the Project We will have a multi-sig wallet for the owner in the future.