Yieldoor Audit Report

Prepared by 0xSimao

Version 2.0

November 30, 2025

Contents

1 About 0xSimao

0xSimao is an indepedent security researcher, #2 on Sherlock, top-ranked on Cantina and Code4rena, and a member of Blackthorn, a leading auditing firm. Previously, he served as Head of Security at Three Sigma.

0xSimao has placed in the Top 3 in 28 public audits and has led over 60 private engagements. For private audits or collaboration opportunities, feel free to reach out to him on X (@0xSimao), Telegram (@OxSimao), or Discord (@0xSimao).

2 Disclaimer

0xSimao makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Yieldoor is a DeFi yield protocol built to make your capital work harder through optimized, automated strategies across liquidity management, lending, and leveraged yield.

We combine some of the most powerful concepts in DeFi—like concentrated liquidity, fixed-yield arbitrage, and capital-efficient leverage—into a seamless platform for both passive and active users.

5 Audit Scope

The scope is the `ShadowStrategyGauge::collectGaugeRewards()` function.

6 Executive Summary

Over the course of 2 days, 0xSimao conducted an audit on the Yieldoor smart contracts provided by Yieldoor. In this period, a total of 8 issues were found.

Summary

Project Name	Yieldoor
Repository	yieldoor
Commit	379e78619a56...
Fix Commit	0a835bbc12b0...
Audit Timeline	May 12th - May 13th
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	5
Informational	2
Gas Optimizations	0
Total Issues	8

Summary of Findings

[M-1] Fees or vested position withdrawal in ShadowStrategyGauge::addVestingPosition() when collecting fees will be revested	Acknowledged
[L-1] DoS risk in IRamsesGauge(gauge).getReward() when enough periods have passed	Acknowledged
[L-2] Newly added rewards will not be immediately collected	Acknowledged
[L-3] ShadowStrategyGauge::addVestingPosition() may revert if there is 0 liquidity to add	Acknowledged
[L-4] If any of the reward tokens doesn't have a liquid pool, it may be impossible to collect rewards	Resolved
[L-5] Swap amount can be 0 as it divides the balance by 2, DoSing ShadowStrategyGauge::collectGaugeRewards()	Acknowledged
[I-1] If enough rebalances are performed, ShadowStrategyGauge::collectGaugeRewards() could be DoSed	Acknowledged
[I-2] If xShadow is token0 or token1, ShadowStrategyGauge::collectGaugeRewards() will take protocol fee on user funds	Acknowledged

7 Findings

7.1 Medium Risk

7.1.1 Fees or vested position withdrawal in ShadowStrategyGauge::addVestingPosition() when collecting fees will be revested

Description: `ShadowStrategyGauge::_addVestingPosition()` collects fees first, coming either from LP fees or withdrawing from the vested position. Any leftover balance exceeding the initial balance after this call in `ShadowStrategyGauge::collectGaugeRewards()` is considered vested position, but this may result from already unvested amounts or claimed fees, which will be vested again.

Recommended Mitigation: Collect fees at the beginning of the ShadowStrategyGauge::collectGaugeRewards() call, before the balance snapshot is taken.

7.2 Low Risk

7.2.1 DoS risk in IRamsesGauge(gauge).getReward() when enough periods have passed

Description: `IRamsesGauge(gauge).getReward()` iterates through all periods since the first period in the Gauge, which means that whenever a new token is added, it will loop through all periods. Thus, given enough time, there is a DoS risk, although it is likely not significant as it would require a lot of time to pass (probably not even in a lifetime).

Recommended Mitigation: The Gauge has a `getPeriodReward()` function which can be used to collect rewards.

7.2.2 Newly added rewards will not be immediately collected

Description: Reward tokens are set on Yieldoor manually, which means there is a window where rewards may be added in the Gauge, but they weren't yet added on Yieldoor.

Recommended Mitigation: Rewards are never lost as they can be claimed next week in the following `ShadowStrategyGauge::collectGaugeRewards()` call, but it can still go a week without claiming them. It may be managed by always verifying the current rewards before calling `ShadowStrategyGauge::collectGaugeRewards()`, but this is not 100% flawless.

7.2.3 ShadowStrategyGauge::addVestingPosition() may revert if there is 0 liquidity to add

Description: `ShadowStrategyGauge::_addVestingPosition()` always tries to mint liquidity, even if it is null. Thus, it may result in reverting, although the chance is extremely low and there isn't any relevant impact as there would be no liquidity to vest anyway (or it is very low).

Recommended Mitigation: Skip adding liquidity if null.

7.2.4 If any of the reward tokens doesn't have a liquid pool, it may be impossible to collect rewards

Description: `ShadowStrategyGauge::collectGaugeRewards()` swaps all rewards to token0 and token1 through a pool. If this pool doesn't exist or has low liquidity, it may be impossible to swap and collect rewards.

Recommended Mitigation: Add an alternative mechanism to handle this situation.

0xSimao:

Fixed in #dc55cc7.

7.2.5 Swap amount can be 0 as it divides the balance by 2, DoSing ShadowStrategyGauge::collectGaugeRewards()

Description: The `ShadowStrategyGauge::collectGaugeRewards()` will be DoSed when the `swapAmount` balance is 1, as it tries to swap in an amount of `swapAmount / 2`, which reverts. This may make it easier for anybody to donate just 1 token of the reward tokens to intentionally make it revert. It can be fixed by donating more tokens, but it's still something to be avoided.

Recommended Mitigation: Skip if the swap amount after dividing by 2 is null.

7.3 Informational

7.3.1 If enough rebalances are performed, ShadowStrategyGauge::collectGaugeRewards() could be DoSed

Description: `ShadowStrategyGauge::collectGaugeRewards()` receives all lower and upper tick positions in the past period. Thus, as rebalances may be performed as soon as every few minutes, it could lead to passing hundreds of positions to collect rewards for all, which could DoS the function call.

Recommended Mitigation: It's unlikely this will happen as rebalancing is not expected to be called this often, but it is something to keep in mind.

7.3.2 If xShadow is token0 or token1, ShadowStrategyGauge::collectGaugeRewards() will take protocol fee on user funds

Description: Although at the moment there isn't an `xShadow` pool, so it's impossible for it to be token0 or token1, `ShadowStrategyGauge::collectGaugeRewards()` would exit `xShadow` to `shadow`, and then convert it back to `xShadow`, taking a protocol fee in the process.

Recommended Mitigation: Add a comment or similar.