GGP-Vault Security Review

AlwaysBeGrowing

Reviewers
Reviewer 1, Ethan Cemer
February 20, 2024

1 Executive Summary

Over the course of 6 days in total, GGP-Vault engaged with Always Be Growing to review GGP-Vault protocol.

We found a total of 4 issues with GGP-Vault.

Repository	Commit
GGP-Vault	aa4758b

Summary

Type of Project	Smart Contract
Timeline	Feb 10, 2024 - Feb 20, 2024
Methods	Manual Review
Documentation	Medium
Testing Coverage	Medium

Total Issues

Critical Risk	0
High Risk	1
Medium Risk	2
Low Risk	1
Gas Optimizations and Informational	0

Contents

2 Always Be Growing

ABG is a talented group of engineers focused on growing the web3 ecosystem. Learn more at https://abg.garden

3 Introduction

This document introduces a vault that adheres to the ERC4626 standard, leveraging the GoGoPool protocol's token, GGP, as its primary asset. The vault employs a strategy that involves staking GGP tokens through trusted node operators to earn yield rewards.

When GGP tokens are deposited, the vault issues receipt tokens, denoted as xGGP, to represent the depositor's stake in the vault's assets. These xGGP tokens reflect the ownership of the vault's assets and are instrumental in the yield generation process. The yield is generated by staking GGP tokens via trusted node operators within the GoGoPool protocol, which rewards this staking activity. As rewards accumulate, they enhance the value of the xGGP tokens, which holders can redeem for their original GGP tokens.

Disclaimer: This security assessment does not provide a guarantee against potential breaches. It represents a focused review based on the code's status at a specific commit and conducted by an individual. Any changes to the code necessitate another comprehensive security evaluation.

4 Architecture

The vault builds upon the core functionalities provided by reputable and audited contracts from the OpenZeppelin library, with modifications to facilitate interaction with the GoGoPool protocol. It is crafted to be upgradeable, which supports the implementation of future enhancements and the correction of any detected flaws.

Consequently, the primary considerations include:

1. Compliance with the ERC-4626 standard by the contract.
2. Security of the modifications made to the OpenZeppelin contracts.
3. Evaluation of potential unecessary centralization risks due to these modifications.

5 Methodology

The following is steps I took to evaluate the change.

- Clone the repo and install dependencies
- Line by line review
  - Contract (including inherited OZ contracts)
  - Interface
  - Unit Tests
  - Integration Tests
- Run ERC-4626 Test Suite
- Write my own set of scenario based tests

6 Findings

6.1 Critical Risk

No findings

6.2 High Risk

1 finding

6.3 Medium Risk

2 findings

6.3.1 `depositYield()` can be front run

Severity: High **Context:** ggpVault.sol#depositYield **Description:** For ggpVault, it seems that yield earned from staking is collected through `depositYield()` callable by the owner or node operator.

However, in the current implementation, uncollected yields are not included in `totalAssets()`, making it vulnerable to front-run attacks that steal pending yields.

Proof of Concept:

Given:

- Current `totalAssets()` is 10,000 GGP;
- Current unclaimed yields (trading fees) is 2,000 GGP;
- Node operator or owner calls `depositYield()` to collect fees and reinvest;
- The attacker sends a deposit tx with a higher gas price to deposit 10,000 GGP, take 50% share of the vault;
- After the transaction in step 1 is packed, the attacker calls `withdraw` and retrieves 10,000 + (0.5)*(1,000) == 10500 GGP.

As a result, the attacker has stolen half of the pending yields in about 1 block of time.

Recommendation:

In general to prevent free loaders from front running yield, Seafi should disable deposits until after all the yield has been deposited back into the vault for the reward cycle. This will also ensure that all the depositors prior capital used for earning rewards get a fair pro rata share of the rewards.

Resolution:

Rather than disable deposits, the `stakeAndDistributeRewards` function takes 100% of the GGP in the vault and then optimistically distributes rewards 9ebc70d.

6.3.2 `distributeRewards` Vulnerable to Front Running

Severity: Medium

Context: ggpVault.sol#distributeRewards

Description: The function `distributeRewards` is exposed as an external function, making it callable by the `owner`. This design choice allows the owner to front-run the `stakeAndDistribute` function call, potentially diverting rewards intended for legitimate stakers, posing undue centralization risk.

Proof of Concept:

- Attacker monitors the transaction pool for `stakeAndDistribute` calls.
- Before the legitimate `stakeAndDistribute` call is confirmed, the attacker submits a `distributeRewards` call with a higher gas price.

- The attacker's transaction is confirmed first, distributing rewards based on the current state before the legitimate update from `stakeAndDistribute`.

Recommendation: Restrict the visibility of `distributeRewards` to internal or only allow it to be called from within `stakeAndDistribute`. This would prevent unauthorized access and ensure that rewards distribution is tightly coupled with the staking process, eliminating the risk of front-running.

Resolution: Seafi team agreed to restrict the visibility of `distributeRewards` to internal only during the next contract upgrade.

6.3.3 `getRewardsBasedOnCurrentStakedAmount` Susceptible to Overflow

Severity: Medium

Context: ggpVault.sol#getRewardsBasedOnCurrentStakedAmount

Description: The function `getRewardsBasedOnCurrentStakedAmount`, which relies on `previewRewardsAtStakedAmount`, is vulnerable to multiplication overflow. This occurs when the vault's cap is reached, and the calculated `targetAPR * GGCap` equals `type(uint256).max`, resulting reverts when `stakeAndDistributeRewards` is called.

Proof of Concept:

- Assume `targetAPR` and `GGCap` are set to values that, when multiplied, exceed `type(uint256).max`.
- The overflow in the multiplication is detected by Solidities built in safe math libarary, resulting reverts for the `previewRewardsAtStakedAmount` function.

Recommendation: Implement checks to ensure that the product of `targetAPR * GGCap` never reaches `type(uint256).max`. Alternatively, use the `mulDiv` library for multiplication, which includes overflow protection and accurate calculations. This approach ensures that rewards are calculated safely and accurately, even at the vault's capacity limit.

Resolution: Right now the current supply of GGP is 20 million, so such an issue is infeasible at the moment. However the supply of GGP will continue to increase so eventually this will become an issue, in which case the Seafi team will upgrade the conracts with the recommend fix.

6.4 Low Risk

1 finding

6.4.1 `stakeAndDistributeRewards` can be DOSed

Severity: low **Context**: ggpVault.sol#stakeAndDistributeRewards **Description**: It is possible for a whale to deposit the maximum amount of tokens, blocking all other deposits and then immediately withdrawing all the capital as soon as Seafi team calls `stakeAndDistributeRewards` .

Recommendation:

To prevent DOS attacks like this, I recommend disabling withdrawals for a period of time before Seafi team calls the `stakeAndDistributeRewards` function.

Resolution: Seafi team decided that although such an attack is possible, it is not economically viable (meaning the attacker would simply lose money in gas fees for executing the attack), so its unlikely to occur. If such an attack where to happen though the Seafi team would upgrade the contract and lock withdrawals until they call the `stakeAndDistributeRewards` function at the end of cycle.

6.5 Informational

0 findings

7 Additional Risks

In addition to the identified security vulnerabilities and their respective resolutions, it is essential to consider the following additional risks that could potentially impact the stability and security of the vault:

7.1 Market Volatility

The value of GGP tokens, like all cryptocurrency assets, is subject to high market volatility. Significant price fluctuations can affect the vault's overall performance and the effectiveness of its staking strategy. Such volatility may lead to unpredictable impacts on the vault's yield generation and collateralization ratios.

7.2 Smart Contract Risk

Even with thorough testing and audits, the inherent complexity of smart contracts can lead to undiscovered vulnerabilities. The use of upgradeable contracts introduces additional layers of complexity and potential points of failure, which could be exploited by malicious actors.

7.3 Protocol Dependency

The vault's operations are heavily dependent on the GoGoPool protocol and its associated contracts. Any updates or changes to the GoGoPool protocol could unpredictably affect the vault's staking strategy and yield generation. Continuous monitoring and adaptability to protocol changes are crucial for maintaining the vault's performance and security.

7.4 Governance and Centralization

The upgradeability of the vault and its reliance on a set of privileged roles for operational decisions introduce centralization risks. The power to upgrade contracts or modify critical parameters could be misused, intentionally or unintentionally, leading to adverse outcomes for the vault and its stakeholders.

7.5 Conclusion

The audit has meticulously evaluated the vault's adherence to the ERC-4626 standard, the effectiveness of its security features, and the security implications of its modifications to the OpenZeppelin contracts. While the audit has identified and addressed several risks of varying severity, it is also important to acknowledge and plan for additional risks related to market volatility, smart contract vulnerabilities, protocol dependencies, and governance centralization.

The proactive resolution of identified issues and the consideration of potential additional risks are indicative of a commitment to maintaining a secure and robust system. It is recommended that continuous monitoring, routine audits before upgrades, and a responsive approach to emerging threats and opportunities be adopted to ensure the long-term success and security of the vault.

In conclusion, while the audit findings present a constructive path toward enhancing the vault's security posture, the dynamic nature of the web3 ecosystem

necessitates ongoing vigilance and adaptability. The implementation of the recommended measures, combined with a proactive risk management strategy, will contribute significantly to the vault's resilience against threats and its overall performance in the Avax ecosystem.