EasyRetroPGF Security Review

Auditors
0xKaden, Security Researcher

March 11, 2024

1 Executive Summary

Over the course of 5 days in total, Allo engaged with 0xKaden to review EasyRPGF-Strategy.sol

Metadata

Repository	Commit
easy-retro-pgf-allo2-strategy	fa2c418

Summary

Type of Project	Public Goods Funding
Timeline	March 6th, 2024 - March 10th, 2024
Methods	Manual Review

Total Issues

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Gas Optimizations	1

Contents

2 Introduction

Allo is an open-source protocol that enables groups to efficiently and transparently allocate pooled capital.

The focus of the security review was on the following:

1. The EasyRPGFStrategy.sol contract

Disclaimer: This review does not make any warranties or guarantees regarding the discovery of all vulnerabilities or issues within the audited smart contracts. The auditor shall not be liable for any damages, claims, or losses incurred from the use of the audited smart contracts.

3 Findings

3.1 Critical Risk

No critical risk findings were discovered.

3.2 High Risk

No high risk findings were discovered.

3.3 Medium Risk

No medium risk findings were discovered.

3.4 Low Risk

3.4.1 Consider reverting unused state changing functions

Severity: Low

Context:

- `EasyRPGFStrategy.sol#L71-L74`
- `EasyRPGFStrategy.sol#L84-L87`

Description:

`allocate` and `registerRecipient` are both payable functions:

```
function allocate(bytes memory _data, address _sender) external payable
 ↪  onlyAllo onlyInitialized {
    _beforeAllocate(_data, _sender);
    _allocate(_data, _sender);
    _afterAllocate(_data, _sender);
}
```

```
function registerRecipient(bytes memory _data, address _sender)
    external
    payable
    onlyAllo
    onlyInitialized
    returns (address recipientId)
{
    _beforeRegisterRecipient(_data, _sender);
    recipientId = _registerRecipient(_data, _sender);
    _afterRegisterRecipient(_data, _sender);
}
```

We define `_allocate` and `_registerRecipient` as noop's:

```
function _allocate(
    bytes memory _data,
    address _sender
) internal virtual override {}
```

```
function _registerRecipient(
    bytes memory _data,
    address _sender
) internal virtual override returns (address) {}
```

In the contract, it's impossible to `distribute` or `withdraw` more pool tokens than are accounted for by `poolAmount`. This is because we decrement `poolAmount` by the amount being transferred, and any underflow will trigger a revert due to Solidity v0.8 SafeMath. Since `allocate` and `registerRecipient` both do not update the `poolAmount`, any time a non-zero `msg.value` is transferred along with a call to either of these functions, that ETH will be permanently locked in the contract. Furthermore, since these functions are noop's, we may as well revert execution to prevent poor user experience from the assumption that the functions executed some logic or state changes.

Recommendation:

Consider immediately reverting in `_allocate` and `_registerRecipient`. **Note that the following code is untested and may contain bugs.**

```
function _allocate(
    bytes memory _data,
    address _sender
- ) internal virtual override {}
+ ) internal virtual override {
+   revert NOOP();
+ }
```

```
function _registerRecipient(
    bytes memory _data,
    address _sender
- ) internal virtual override returns (address) {}
+ ) internal virtual override returns (address) {
+   revert NOOP();
+ }
```

Allo: Fixed in #0991699.

0xKaden: Resolved.

3.4.2 Lack of NatSpec documentation

Severity: Low

Context: `EasyRPGFStrategy.sol#L8`

Description:

NatSpec documentation is valuable both for developers reading the code as well as users interacting with the contracts via external applications like Etherscan. Providing a clear understanding to developers and users is highly valuable. As such, NatSpec documentation should be implemented in the contract as described in the Solidity documentation.

Allo: Fixed in #e69969d.

0xKaden: Resolved.

3.4.3 Unused library import

Severity: Low

Context: `EasyRPGFStrategy.sol#L5`

Description:

The Metadata library is imported yet unused:

```
import {IAllo, Metadata} from "allo-v2/contracts/core/interfaces/IAllo.sol";
```

This import should be removed to reduce redundancy, and improve overall code quality.

Allo: Fixed in #1f4ff89.

0xKaden: Resolved.

3.5 Gas Optimizations

3.5.1 Efficient reentrancy protection

Severity: Gas

Context: `EasyRPGFStrategy.sol#L26`

Description:

`withdraw` uses the `nonReentrant` modifier from `ReentrancyGuard` to prevent reentrancy:

```
function withdraw(
    address _recipient
) external nonReentrant onlyPoolManager(msg.sender) {
    IAllo.Pool memory pool = allo.getPool(poolId);
    _transferAmount(pool.token, _recipient, poolAmount);
    poolAmount = 0;
}
```

However, we can much more efficiently prevent reentrancy simply by following the checks-effects-interactions pattern. We can do this by simply setting the `poolAmount` to 0 prior to transferring the tokens. Since all of our state changes (effects) would have been made by the time we execute this external call (interactions), reentrancy is no longer a concern.

This allows us to remove the `nonReentrant` modifier and not inherit the `ReentrancyGuard` contract. The result of this is an estimated ~40000 gas saved on deployment cost by reducing bytecode size and ~2300 gas saved per `withdraw` call.

Recommendation:

Remove the `nonReentrant` modifier and `ReentrancyGuard` inheritance, and reset the `poolAmount` before executing `_transferAmount`. **Note that the following code is untested and may contain bugs.**

```
function withdraw(
    address _recipient
- ) external nonReentrant onlyPoolManager(msg.sender) {
+ ) external onlyPoolManager(msg.sender) {
    IAllo.Pool memory pool = allo.getPool(poolId);
+   uint256 _poolAmount = poolAmount;
+   poolAmount = 0;
-   _transferAmount(pool.token, _recipient, poolAmount);
+   _transferAmount(pool.token, _recipient, _poolAmount);
-   poolAmount = 0;
}
```

Allo: Fixed in #62dec5a.

0xKaden: Resolved.