DirectGrantsLite Security Review

Auditors
0xKaden, Security Researcher

May 6, 2024

1 Executive Summary

Over the course of 3 days in total, Allo engaged with 0xKaden to review DirectGrantsLite.

Metadata

Repository	Commit
direct-grants-lite	e5c40ca

Summary

Type of Project	Public Goods Funding
Timeline	May 2nd, 2024 - May 5th, 2024
Methods	Manual Review

Total Issues

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

Contents

1 Executive Summary **1**

2 Introduction **3**

3 Findings **3**

3.1 Critical Risk . . . 3
3.2 High Risk . . . 3
3.3 Medium Risk . . . 3
3.3.1 `reviewRecipients` can be DoS'd . . . 3
3.3.2 Recipients can frontrun `reviewRecipients` to update their recipient data . 4
3.4 Low Risk . . . 4
3.4.1 Prevent unexpected msg.value in `_registerRecipient` . . . 4
3.4.2 Unsafe `receive` function . . . 5
3.5 Informational . . . 5
3.5.1 Typos and misleading comments . . . 5
3.5.2 Add revert messages . . . 6
3.6 Gas Optimizations . . . 6
3.6.1 Reference parameters instead of storage . . . 6
3.6.2 Use unchecked loop incrementor . . . 7

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 `DirectGrantsLite.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

3.3.1 `reviewRecipients` can be DoS'd

Severity: Medium

Context: DirectGrantsLite.sol#L252

Description:

In the `reviewRecipients` function, the pool manager must pass a `refRecipientsCounter` parameter which must be equal to the actual `recipientsCounter` storage variable which is incremented every time a new recipient is registered:

```
if (refRecipientsCounter != recipientsCounter) revert INVALID();
```

This is done to prevent a possible race condition wherein a `_registerRecipient` transaction is processed shortly before a `reviewRecipients` transaction is executed which contains now obsolete data for that recipient. Failing to prevent this race condition can result in the recipient registration being accidentally overwritten.

The tradeoff that this logic is making, however, is that it allows for an attacker to DoS this function by frontrunning its execution with calls to `_registerRecipient` from new accounts. The attacker can always ensure that `refRecipientsCounter != recipientsCounter`, causing `reviewRecipients` to always fail. Since `reviewRecipients` must be executed for allocation to be possible, this fundamentally compromises the functionality of the contract.

Recommendation:

This can be prevented in a couple different ways. One option is to only allow `reviewRecipients` to be executed after the registration period is completed. This simplifies the process by preventing overlap between the execution windows of `_registerRecipient` and `reviewRecipients`. Another option is to apply updated recipient statuses individually on to the current `statusesBitMap` `rowIndex` instead of updating the entire row regardless of any recent status changes.

3.3.2 Recipients can frontrun `reviewRecipients` to update their recipient data

Severity: Medium

Context: DirectGrantsLite.sol#L349

Description:

In `_registerRecipient`, recipients provide the following data for review by the pool manager: `recipientAddress`, `metadata`, and `useRegistryAnchor`. Recipients can update this data at any time in the registration period by calling the function again with new data. This allows for attackers to frontrun `reviewRecipients` by calling `_registerRecipient` again to arbitrarily modify the aforementioned data. The result of this attack is that the pool manager accepts a recipient with data which they may not have otherwise accepted.

Recommendation:

Similarly to 3.3.1, we can prevent this operationally by only allowing `reviewRecipients` to be called after the registration period is complete, preventing overlap between these two functions. Alternatively, we can prevent this by including a hash of the expected state of each recipient to be verified against the current state of the recipient.

3.4 Low Risk

3.4.1 Prevent unexpected msg.value in `_registerRecipient`

Severity: Low

Context: DirectGrantsLite.sol#L349

Description:

`_registerRecipient` is executed via `BaseStrategy.registerRecipient`, a `payable` function which users must call via `Allo.sol`, with the option to provide native tokens along with the call. This is to support use cases of `registerRecipient` in other Allo strategies but is not used by `DirectGrantsLite`. Any `msg.value` passed to `_registerRecipient` gets left in the contract and can only be withdrawn by the pool manager.

Recommendation:

To prevent native tokens from being unexpectedly transferred to the contract in `_registerRecipient`, we can include a check which reverts if the `msg.value` is non-zero, e.g.:

```
if (msg.value != 0) revert NON_ZERO_VALUE();
```

3.4.2 Unsafe `receive` function

Severity: Low

Context: DirectGrantsLite.sol#L535

Description:

`DirectGrantsLite` contains a receive function, allowing native tokens to be transferred directly to the contract. As described in 3.4.1, native tokens are not expected to be transferred to the contract and can only be removed by the pool manager.

Recommendation:

Remove the `receive` function from the contract to prevent native tokens from being unexpectedly transferred to the contract.

3.5 Informational

3.5.1 Typos and misleading comments

Severity: Informational

Context:

- DirectGrantsLite.sol#L29
- DirectGrantsLite.sol#L136
- DirectGrantsLite.sol#L297
- DirectGrantsLite.sol#L38-L40
- DirectGrantsLite.sol#L234-L236
- DirectGrantsLite.sol#L125-L126

Description:

The following typos and misleading comments are present in the codebase:

- L29: `/// @title DDirect Grants Lite Strategy`, should be: `/// @title Direct Grants Lite Strategy`
- L136: `an recipient`, should be: `a recipient`
- L297: `uint256 accessableAmount = amount;`, should be: `uint256 accessibleAmount = amount;`
- L38-L40 & L234-L236: These comments indicate the ordering of statuses in an unclear manner, "The first 4 bits... represent... the first recipient" intuitively indicates that the stasuses will be ordered from left to right when in reality they are ordered from right to left. This can be better clarified to avoid confusion and improve readability.
- L125-L126: Here we indicate that 5 different statuses are used when in reality there are 7 possible statuses.

Recommendation:

Update the noted lines as indicated above.

3.5.2 Add revert messages

Severity: Informational

Context:

- DirectGrantsLite.sol#L426
- DirectGrantsLite.sol#L490

Description:

Both `_distribute` and `_getPayout` are functions which immediately revert because they're unused for this strategy. In both instances, there is no revert message which can be useful to indicate to the user why the function reverted.

Recommendation:

Add revert messages indicating why the functions are reverting.

3.6 Gas Optimizations

3.6.1 Reference parameters instead of storage

Severity: Gas optimization

Context:

- DirectGrantsLite.sol#L203-L206
- DirectGrantsLite.sol#L287

Description:

In `__DirectGrantsLiteStrategy_init` and `updatePoolTimestamps`, we update the storage variables `registrationStartTime` and `registrationEndTime` by setting them as the function parameters, e.g. in `updatePoolTimestamps`:

```
registrationStartTime = _registrationStartTime;
registrationEndTime = _registrationEndTime;
```

In these functions, we then use the storage variables instead of the parameters to reference these values, e.g. in `updatePoolTimestamps`:

```
emit TimestampsUpdated(registrationStartTime, registrationEndTime, msg.sender);
```

Recommendation:

Since we have access to the parameters used to set the storage variables, and accessing storage variables is an expensive operation, we can simply use the parameters directly, e.g. in `updatePoolTimstamps` we can do:

```
-emit TimestampsUpdated(registrationStartTime, registrationEndTime, msg.sender);
+emit TimestampsUpdated(_registrationStartTime, _registrationEndTime, msg.sender);
```

In `__DirectGrantsLiteStrategy_init`, we can also pass the parameters to `_isPoolTimestampValid`, e.g.:

```
// If the timestamps are invalid this will revert - See details in
↪  '_isPoolTimestampValid'
-_isPoolTimestampValid(registrationStartTime, registrationEndTime);
+_isPoolTimestampValid(_initializeData.registrationStartTime,
↪  _initializeData.registrationEndTime);

// Emit that the timestamps have been updated with the updated values
-emit TimestampsUpdated(registrationStartTime, registrationEndTime, msg.sender);
+emit TimestampsUpdated(_initializeData.registrationStartTime,
↪  _initializeData.registrationEndTime, msg.sender);
```

3.6.2 Use unchecked loop incrementor

Severity: Gas optimization

Context: DirectGrantsLite.sol#L443

Description:

In `_allocate`, we have a `for` loop which increments `i` as long as it's less than `length`:

```
for (uint256 i = 0; i < length; i++) {
```

We can assert that `i` will never overflow because `length` would have to be `> type(uint256).max` which is not possible. Knowing this, we can uncheck the incrementor.

Recommendation:

Remove the incrementor from its original position and wrap it in an `unchecked` block at the end of the `for` loop. Note that we can also use `++i` instead of `i++` to save an additional 5 gas per loop in case the optimizer is not used.

```
unchecked {
    ++i;
}
```

Note that using a compiler version of `0.8.22` or higher will automatically uncheck the incrementor when safe to do so, although before making this change it should be carefully considered whether the networks you intend to deploy on support this version.