DirectGrantsSimpleStrategy and Permit PR Security Review

Auditors
0xKaden, Security Researcher

March 17, 2024

1 Executive Summary

Over the course of 7 days in total, Allo engaged with 0xKaden to review DirectGrantsSimpleStrategy.sol and Permit support PR.

Metadata

Repository	Commit
direct-grants-simple	6b3ad15
Pull Request 472	9b082a7

Summary

Type of Project	Public Goods Funding
Timeline	March 4th, 2024 - March 10th, 2024
Methods	Manual Review

Total Issues

Critical Risk	0
High Risk	0
Medium Risk	4
Low Risk	10
Gas Optimizations	5

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 DirectGrantsSimpleStrategy.sol contract
2. Pull request #472 - "Permit and DAI permit support"

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 Recipients can frontrun reviewSetMilestones with setMilestones to have unexpected milestones accepted

Severity: Medium

Context: `DirectGrantsSimpleStrategy.sol#L299-L320`

Description:

`setMilestones` can be called multiple times as long as `milestonesReviewStatus != Status.Accepted`:

```
if (recipient.milestonesReviewStatus == Status.Accepted) {
    revert MILESTONES_ALREADY_SET();
}
```

When the `poolManager` calls `reviewSetMilestones`, they have no way of enforcing that the milestones they're accepting are the actual milestones currently in the contract state. Instead, it simply sets the current milestones as `Status.Accepted`:

```
if (_status == Status.Accepted || _status == Status.Rejected) {
    // Set the status of the milestone review
    recipient.milestonesReviewStatus = _status;

    // Emit event for the milestone review
    emit MilestonesReviewed(_recipientId, _status);
}
```

A malicious recipient can exploit this functionality as follows:

- The attacker initially sets reasonable milestones using `setMilestones`
- The attacker watches the mempool for a call from the `poolManager` to review their milestones as `Status.Accepted`
- The attacker frontruns this call by setting unreasonable milestones using `setMilestones`
- The unreasonable milestones are accepted

Most significantly, the attacker can modify the share of `milestone.amountPercentage` from a reasonable, even distribution to an unreasonable distribution in which the initial milestone has an `amountPercentage` of 1e18 (100%).

Ultimately, assuming the `poolManager` uses the contract state as the single source of truth, this may result in the `poolManager` distributing funds inconsistently with how they initially approved.

Recommendation:

`reviewSetMilestones` should include a `milestonesHash` parameter for the `poolManager` to provide, such that if the milestones have changed, execution will revert. **Note that the following is untested and may contain bugs.**

```
- function reviewSetMilestones(address _recipientId, Status _status) external
 ↪  onlyPoolManager(msg.sender) {
+ function reviewSetMilestones(address _recipientId, Status _status, bytes32
 ↪  milestonesHash) external onlyPoolManager(msg.sender) {
    Recipient storage recipient = _recipients[_recipientId];

+   if (keccak256(abi.encode(milestones[_recipientId])) != milestonesHash)
 ↪  revert INVALID_MILESTONES();
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.3.2 Lack of validation may allow attacker to reenter during distribution to arbitrarily modify milestones

Severity: Medium

Context: `DirectGrantsSimpleStrategy.sol#L328`

Description:

Both `submitMilestone` and `_distributeUpcomingMilestone` fail to validate that recipients milestones have been reviewed. `submitMilestone` doesn't validate `recipient.milestonesReviewStatus`, allowing unreviewed milestones to be submitted. Nor does `_distributeUpcomingMilestone`, allowing unreviewed milestones to be distributed. This lack of validation devalues the review process by allowing the remainder of the milestone lifecycle to proceed regardless.

In the worst case, if multiple unreviewed milestones are being distributed for the same malicious recipient, the recipient can perform a reentrancy attack (if ETH is the pool token) to receive more than their entire allotted `grantAmount`. The attack works as follows:

- Attacker sets reasonable milestones with `setMilestones`
- Attacker calls `submitMilestone` for each milestone, bypassing the review process
- `poolManager` mistakenly distributes for each attacker milestone under the assumption that since the milestones are reasonable and have been submitted that they've been reviewed
- Attacker frontruns distribution by resetting the milestones using a similar

attack described in 3.3.1 to set the first milestone with an `amountPercentage` of 1e18 (100%)
  - This is possible because `recipient.milestonesReviewStatus != Status.Accepted` since it hasn't been reviewed
- When the ETH is transferred to the attacker, they can reenter to reset their milestones again such that the next indexed milestone also sets `grantAmount` to 1e18 (100%)
- Attacker can repeat this process for each milestone, receiving multiples of their intended `grantAmount`

Listing this as medium severity since it requires a lack of diligence on the part of the `poolManager`, yet the impact is high with a loss of funds which are allocated for other recipients.

Recommendation:

Enforce that `recipient.milestonesReviewStatus == Status.Accepted` in `submitMilestone`. Since `_distributeUpcomingMilestone` will revert if `milestoneStatus != Status.Pending`, this effectively prevents distribution of unreviewed milestones. **Note that the following is untested and may contain bugs.**

```
function submitMilestone(address _recipientId, uint256 _milestoneId, Metadata
 ↪  calldata _metadata) external {
    // Check if the '_recipientId' is the same as 'msg.sender' and if it is
     ↪  NOT, revert. This
    // also checks if the '_recipientId' is a member of the 'Profile' and if it
     ↪  is NOT, revert.
    if (_recipientId != msg.sender && !_isProfileMember(_recipientId,
     ↪  msg.sender)) {
        revert UNAUTHORIZED();
    }

    Recipient memory recipient = _recipients[_recipientId];

+   if (recipient.milestonesReviewStatus != Status.Accepted) revert
 ↪  MILESTONES_NOT_ACCEPTED()
```

Allo: Acknowledged, but will not fix.

0xKaden: Acknowledged.

3.3.3 Permit `catch` checks the allowance of the wrong account

Severity: Medium

Context:

- `DonationVotingMerkleDistributionDirectTransferStrategy.sol#L82`
- `DonationVotingMerkleDistributionDirectTransferStrategy.sol#L86`
- `DonationVotingMerkleDistributionDirectTransferStrategy.sol#L97`
- `DonationVotingMerkleDistributionDirectTransferStrategy.sol#L101`

Description:

In `DonationVotingMerkleDistributionDirectTransferStrategy._afterAllocate`, if the `permit` call fails, we fallback to check the allowance and only revert if the allowance is also insufficient to make the transfer:

```
try IERC20Permit(token).permit(_sender, address(this), amount,
 ↪  p2Data.permit.deadline, v, r, s) {}
catch Error(string memory reason) {
    if (IERC20(token).allowance(msg.sender, address(this)) < amount) {
        revert(reason);
    }
} catch (bytes memory reason) {
    if (IERC20(token).allowance(msg.sender, address(this)) < amount) {
        revert(string(reason));
    }
}
```

However, when checking the allowance, we make the mistake of passing `msg.sender` as the `owner` param while the actual sender which we intend to check the allowance of is `_sender`. This causes the allowance check to be performed on the `allo` contract, since `allocate` may only be called by `allo`.

As a result, DoS attacks via frontrunning permit will successfully block execution, regardless of this intended prevention mechanism.

Recommendation:

Replace `msg.sender` with `_sender` for each allowance call. **Note that there are four different allowance calls to be fixed.**

```
- if (IERC20(token).allowance(msg.sender, address(this)) < amount) {
+ if (IERC20(token).allowance(_sender, address(this)) < amount) {
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.3.4 Recipients cannot be marked as `InReview` and then later re-accepted without breaking an invariant

Severity: Medium

Context: DirectGrantsSimpleStrategy.sol#L393-L405

Description:

The only way to mark a recipient's status as `Status.Accepted` is via `_allocate`:

```
recipient.recipientStatus = Status.Accepted;
```

Thus, when a recipient's status is set to `InReview` by `setRecipientStatusToIn-Review`, the only way to mark their status as `Accepted` again is to call `allocate` on that recipient again. The problem with this is that to `allocate` to the recipient again, we must increment `allocatedGrantAmount` by the `grantAmount` being reassigned to the recipient.

```
// @audit allocatedGrantAmount is incremented by grantAmount
allocatedGrantAmount += grantAmount;

// Check if the allocated grant amount exceeds the pool amount and reverts if
↪  it does
if (allocatedGrantAmount > poolAmount) {
    revert ALLOCATION_EXCEEDS_POOL_AMOUNT();
}

// @audit recipient.grantAmount is set as grantAmount
recipient.grantAmount = grantAmount;
recipient.recipientStatus = Status.Accepted;
```

This breaks the invariant that `allocatedGrantAmount` is equal to the sum of the `grantAmount`'s of all `Accepted` recipients. The result of this broken invariant is that the pool must be overfunded since `_allocate` enforces that `allocated-GrantAmount <= poolAmount`.

Recommendation:

To fix this, in `setRecipientStatusToInReview`, for each recipient, we must not only update their `recipientStatus`, but also set their `grantAmount` to 0 and decrement the `allocatedGrantAmount` accordingly. **Note that the following is untested and may contain bugs.**

```
function setRecipientStatusToInReview(address[] calldata _recipientIds)
 ↪  external onlyPoolManager(msg.sender) {
    uint256 recipientLength = _recipientIds.length;
    for (uint256 i; i < recipientLength;) {
        address recipientId = _recipientIds[i];
        _recipients[recipientId].recipientStatus = Status.InReview;
+       allocatedGrantAmount -= _recipients[recipientId].grantAmount;
+       _recipients[recipientId].grantAmount = 0;
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4 Low Risk

3.4.1 Typos, misleading and inaccurate comments

Severity: Low

Context:

- `DirectGrantsSimpleStrategy.sol#L307`
- `DirectGrantsSimpleStrategy.sol#L312`
- `DirectGrantsSimpleStrategy.sol#L323`
- `DirectGrantsSimpleStrategy.sol#L329`
- `DirectGrantsSimpleStrategy.sol#L351`
- `DirectGrantsSimpleStrategy.sol#L511`
- `DirectGrantsSimpleStrategy.sol#L663`

Description:

The following comments are either misleading, inaccurate, or contain typos:

- L307: `// Check if the recipient is 'Accepted', otherwise revert`, should be `// Check if the recipient is not 'Accepted', otherwise revert`.
- L312: `// Check if the status is 'Accepted' or 'Rejected', otherwise revert`, execution doesn't revert if not accepted or rejected.

- L323: `/// @dev 'msg.sender' must be the 'recipientId' (this depends on whether your using registry gating)...`, should be `you're` not `your`.
- L329: `// Check if the '_recipientId' is the same as 'msg.sender' and if it is NOT, revert. This // also checks if the '_recipientId' is a member of the 'Profile' and if it is NOT, revert.`, technically not correct, could be better worded.
- L351: `// Check if the milestone is accepted, otherwise revert`, should be `// Check if the milestone is not accepted, otherwise revert`.
- L511: `// Add the recipient to the accepted recipient ids mapping`, misleading as the recipient is not actually accepted yet.
- L663: `// Reverts if the milestone status is 'None'`, should be `// Reverts if the milestone status is not 'None'`.

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.2 Off-by-one errors in validating `_milestoneId` is a valid milestone

Severity: Low

Context:

- DirectGrantsSimpleStrategy.sol#L345
- DirectGrantsSimpleStrategy.sol#L372
- DirectGrantsSimpleStrategy.sol#L599

Description:

An incorrect logical pattern is used a few times throughout the codebase to enforce that a given milestone index exists in the `milestones[_recipientId]` array:

```
if (_milestoneId > recipientMilestones.length) {
    revert INVALID_MILESTONE();
}
```

This fails to revert in the circumstance where `_milestoneId == recipientMilestones.length`, which is also an invalid index since `recipientMilestones` is 0-indexed.

Luckily this doesn't provide a significant threat as solidity will `Panic` if we try to access an out of bounds index. However, this does prevent us from receiving our intended revert reason: `INVALID_MILESTONE`.

Recommendation:

For each circumstance listed in **Context** above, replace with the following pattern.

```
- if (_milestoneId > recipientMilestones.length) {
+ if (_milestoneId >= recipientMilestones.length) {
    revert INVALID_MILESTONE();
}
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.3 `setPoolActive` emits the `PoolActive` event twice per call

Severity: Low

Context: `DirectGrantsSimpleStrategy.sol#L412`

Description:

In `setPoolActive`, we run the internal `_setPoolActive` function and then `emit` the event:

```
function setPoolActive(bool _flag) external onlyPoolManager(msg.sender) {
    _setPoolActive(_flag);
    emit PoolActive(_flag);
}
```

However, in `_setPoolActive`, we also `emit` the event:

```
function _setPoolActive(bool _active) internal {
    poolActive = _active;
    emit PoolActive(_active);
}
```

Recommendation:

Remove the `emit` from the external function so that it only gets logged once:

```
function setPoolActive(bool _flag) external onlyPoolManager(msg.sender) {
    _setPoolActive(_flag);
-   emit PoolActive(_flag);
}
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.4 Unsafe `receive` function

Severity: Low

Context: `DirectGrantsSimpleStrategy.sol#L696`

Description:

A `receive` function is included in the `DirectGrantsSimpleStrategy` contract:

```
/// @notice This contract should be able to receive native token
receive() external payable {}
```

However, there is no circumstance in which we would want the contract to receive native tokens directly. This is due to the fact that we can only distribute or withdraw the pool token up to the `poolAmount`. So even if the pool token is native, we still have to increment `poolAmount` by the same amount which is being transferred to the contract, otherwise they are permanently locked in the contract.

Recommendation:

Remove the `receive` function.

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.5 TODOs in code

Severity: Low

Context:

- `DirectGrantsSimpleStrategy.sol#L210`
- `DirectGrantsSimpleStrategy.sol#L668`

Description:

There are `TODO` comments present in the code for which the corresponding instructions appear to be complete.

Recommendation:

Ensure the provided instructions have in fact been completed, then remove the `TODO`'s as listed in **Context**.

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.6 Possible to frontun `allocate` by calling `registerRecipient` to unexpectedly modify the recipient metadata

Severity: Low

Context: `DirectGrantsSimpleStrategy.sol#L449`

Description:

When the `poolManager` `allocate`'s to a recipient, it will set their `recipientStatus` as `Status.Accepted` along with the `metadata` provided during registration. However, there exists no logic to validate that the `metadata` initially provided is still present at the time of allocation. It's therefore possible for a recipient to frontrun `allocate` by calling `registerRecipient` again and changing their `metadata`.

Recommendation:

Assuming `metadata` is purely informational, this doesn't appear to cause significant harm, and may be left as an acceptable risk. However, if this is undesirable, it can be fixed by adding a subparam to be decoded in `_data` which contains the hash of the `metadata` to be verified against the current metadata. **Note that the following is untested and may contain bugs.**

```
- (address recipientId, Status recipientStatus, uint256 grantAmount) =
-   abi.decode(_data, (address, Status, uint256));
+ (address recipientId, Status recipientStatus, uint256 grantAmount, bytes32
 ↪  metadataHash) =
+   abi.decode(_data, (address, Status, uint256, bytes32));

Recipient storage recipient = _recipients[recipientId];

+ if (metadataHash != keccak256(abi.encode(recipient.metadata))) revert
 ↪  INVALID_METADATA();
```

Allo: Acknowledged, but will not fix.

0xKaden: Acknowledged.

3.4.7 Section containing structs is incorrectly labelled as `Storage`

Severity: Low

Context: DirectGrantsSimpleStrategy.sol#L37

Description:

Above the section of the contract containing structs, we incorrectly label the section as `Storage`:

```
/// ================================
/// ========== Storage =============
/// ================================
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.8 Permit `try/catch` logic prevents Flashbots Protect transactions from executing

Severity: Low

Context:

- DonationVotingMerkleDistributionDirectTransferStrategy.sol#L80-L89
- DonationVotingMerkleDistributionDirectTransferStrategy.sol#L95-L104

Description:

Permit logic handles the possibility that `permit` has been frontrun, causing execution to revert, by using `try/catch` blocks where in the catch it will only revert if the `allowance` is insufficient to execute the transfer. The problem with this pattern, however, is that the existence of the failed inner call may cause problems with external infrastructure.

As noted in a review comment, this may cause confusion on Etherscan when the transaction is marked as failed.

Additionally, it will prevent Flashbots Protect transactions from being executed, see: "your transaction will only be included if it doesn't include any reverts, so you don't pay for failed transactions.".

Furthermore, there may be additional effects of this pattern which have not yet been considered.

Recommendation:

Note that although the pattern used here is imperfect due to Solidity's flawed error handling, it may still be the best option, particularly since it is the pattern recommended by both OpenZeppelin and Trust Security.

A possible solution may be to include this pattern along with an alternative option to simply execute the transfer as usual under the assumption that `allowance` has already been set, as recommended in **3.5.5**.

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.9 Enforce `msg.value == 0` on payable functions which don't expect ETH

Severity: Low

Context:

- `DirectGrantsSimpleStrategy.sol#L449`
- `DirectGrantsSimpleStrategy.sol#L523`

Description:

As inherited by the base strategy, both `allocate` and `registerRecipient` are payable functions. However, in both instances, passing ETH along with calls to these functions is unsafe because even if the pool token is ETH, we don't increment the `poolAmount` correspondingly and thus can never `distribute` or `withdraw` these funds.

Recommendation:

In `_allocate` and `_registerRecipient`, we should revert if the `msg.value` is non-zero:

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

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.4.10 Unused state variable `_acceptedRecipientIds`

Severity: Low

Context: `DirectGrantsSimpleStrategy.sol#L129`

Description:

The `_acceptedRecipientIds` state variable is unused and should be removed.

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.5 Gas Optimizations

3.5.1 Use `++i` in place of `i++`

Severity: Gas Optimization

Context:

- `DirectGrantsSimpleStrategy.sol#L402`
- `DirectGrantsSimpleStrategy.sol#L582`
- `DirectGrantsSimpleStrategy.sol#L676`

Description:

When incrementing the loop index, it's slightly more gas efficient to do so using `++i` rather than `i++`.

Recommendation:

Replace instances of `i++` as noted in **Context** with `++i`

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.5.2 Redundant milestone existence check

Severity: Gas Optimization

Context: `DirectGrantsSimpleStrategy.sol#L599-L601`

Description:

In `_distributeUpcomingMilestone`, we have the following check:

```
if (milestoneToBeDistributed > recipientMilestones.length ||
 ↪  milestone.milestoneStatus != Status.Pending) {
    revert INVALID_MILESTONE();
}
```

The first part, `milestoneToBeDistributed > recipientMilestones.length`, is redundant because `milestone.milestoneStatus` can't possibly be `Status.Pending` if the index is not included in the array.

Recommendation:

Simplify the check to only include the second part:

```
- if (milestoneToBeDistributed > recipientMilestones.length ||
 ↪  milestone.milestoneStatus != Status.Pending) {
+ if (milestone.milestoneStatus != Status.Pending) {
    revert INVALID_MILESTONE();
}
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.5.3 Save a storage slot by packing storage booleans with `registrationStartTime` and `registrationEndTime`

Severity: Gas Optimization

Context: `DirectGrantsSimpleStrategy.sol#L109-L120`

Description:

`registrationStartTime` and `registrationEndTime` are both 128 bit unsigned integers. Since EVM storage slots contain 256 bits, these two state vars are packed into a single storage slot.

However, 128 bits is overkill for timestamps as it allows us to store a timestamp >1e31 years into the future. Instead, since we have some boolean state vars below which only require 8 bits each, we can make `registrationStartTime` and `registrationEndTime` a bit smaller to pack all these values into a single storage slot. This saves gas on initialization as well as any execution which requires at least one of the listed storage booleans as well as at least one of the timestamps.

Recommendation:

Set `registrationStartTime` and `registrationEndTime` as `uint112`'s since that will still be >1.6e26 years into the future.

```
/// @notice The timestamps in seconds for the start and end times.
- uint128 public registrationStartTime;
+ uint112 public registrationStartTime;
- uint128 public registrationEndTime;
+ uint112 public registrationStartTime;

/// @notice Flag to check if registry gating is enabled.
bool public registryGating;

/// @notice Flag to check if metadata is required.
bool public metadataRequired;

/// @notice Flag to check if grant amount is required.
bool public grantAmountRequired;
```

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.5.4 Consider supporting `EIP-2098` compact signatures

Severity: Gas Optimization

Context: `DonationVotingMerkleDistributionDirectTransferStrategy.sol#L109-L131`

Description:

Current permit signature logic requires that signatures are 65 bytes long:

```
require(sig.length == 65, "invalid signature length");
```

However, as defined in EIP-2098, it's possible to represent the same signature using just 64 bytes, thereby removing the need for a third memory slot.

Recommendation:

Retrieve v, r, and s values as is done by permit2: `https://github.com/Uniswap/permit2/blob/cc56ad0f3439c502c246fc5cfcc3db92bb8b7219/src/libraries/Signatur` `sol#L27-L38`

Allo: Fixed in PR #516.

0xKaden: Resolved.

3.5.5 Consider adding support to simply enable transfers if allowance has already been set

Severity: Gas Optimization

Context: `DonationVotingMerkleDistributionDirectTransferStrategy.sol#L62-L106`

Description:

In `_afterAllocate`, we handle multiple different types of permit's: permit2, EIP-2612 permit, and DAI token permit. What we don't handle, however, is a gas efficient path for simply transferring the tokens that have already been permitted.

This would be useful in the case that the sender:

- Permits enough to cover multiple transfers (e.g. `type(uint256).max`)
- Permits DAI (automatically permits `type(uint256).max`)

Recommendation:

Add a 4th `PermitType`:

```
enum PermitType {
    Permit,
    PermitDAI,
    Permit2,
    Transfer
}
```

And simply execute the transfer if `PermitType.Transfer` is selected:

```
} else if (permitType == PermitType.PermitDAI) {
    (bytes32 r, bytes32 s, uint8 v) = splitSignature(p2Data.signature);
    // The tx can be front-run, and another user can use the permit message and
     ↪  signature to invalidate the nonce.
    // In this case the permit call will fail, but it means that the contract
     ↪  already has allowance for the token.
    try IDAI(token).permit(_sender, address(this), p2Data.permit.nonce,
     ↪  p2Data.permit.deadline, true, v, r, s) {}
    catch Error(string memory reason) {
        if (IERC20(token).allowance(msg.sender, address(this)) < amount) {
            revert(reason);
        }
    } catch (bytes memory reason) {
        if (IERC20(token).allowance(msg.sender, address(this)) < amount) {
            revert(string(reason));
        }
    }
    IERC20(token).transferFrom(_sender,
     ↪  _recipients[recipientId].recipientAddress, amount);
- }
+ } else if (permitType == PermitType.Transfer) {
+   IERC20(token).transferFrom(_sender,
 ↪  _recipients[recipientId].recipientAddress, amount);
+ }
}
```

Note that the above code is untested and may contain bugs.

Allo: Fixed in PR #516.

0xKaden: Resolved.