Context and problem statement

Easy Track is a Lido governance mechanism to ease operational burden for Lido DAO members and it's proved to be efficient and secure.

Right now there are three programs that use Easy Track motions for payouts on regular basis, sourced from the Lido DAO treasury:

- LEGO program;
- ReWARDs program;
- Referral program.

Each program is managed by a designated committee, approved by Lido DAO, and have an off-chain agreement on payouts purpose, limits, and budgets available, e.g. ReWARDS committee proposal.

Managing government processes through committees has proven to be an efficient way of working. The committees, within the scope of their authority, can quickly and efficiently perform operations.

To make the payment, a committee creates a Motion, than waits 72 hours for any objections. If there are no objections, a committee enacts the Motion. And then a transfer transaction fires.

It seems to be productive if the number of committees for different tasks increases over time.

Problem statement

The issue of safety becomes essential. It is necessary to have on-chain limits on the total amount that each of the committees can spend in a certain period.

There is an already implemented per transfer safety cap that limits amounts for each payment initiated by Easy Track.

Now it's time to implement a check so that the Committees do not go beyond their budgets for the period.

If the budget is run out and operations are still to be continued, the committee should post the proposal on the Research forum. Replenishment of the budget beyond the initially set level should be discussed by the community and after that a Snapshot vote should be started.

The Committee cannot change limits and budgets itself. Only DAO can change budget parameters. So it is necessary to make methods for changing budget parameters, for example, its period, or amount available per period. The rights to change parameters must be given to the DAO, for example, Aragon Voting or Aragon Agent.

The ADR focuses on on-chain implementation design of such constraints to improve overall governance transparency, security and make Lido processes more reliable.

Technical overview

Easy Track contract is the main contract, which implements the Easy Track voting mechanics.

During the creation and during the execution of the Motion, EasyTrack contract uses the EVMScriptFactory

contracts to create EVMScripts. EVMScriptFactory

- is a special contract, which implements the IEVMScriptFactory

interface.

Every committee has it's own EVMScriptFactory

instance to make payments.

Motions don't store EVMScripts on-chain but only keccak256

hash of it. On enactment of the motion, EasyTrack contract through the EVMScriptFactory

recreates EVMScript and compares the hash of recreated EVMScript with the stored hash.

[

1604×910 137 KB

](https://europe1.discourse-cdn.com/business20/uploads/lido/original/2X/4/42138be858508bc137acaf013abede8c36a1b43f.jpeg)

Execution of EVMScripts is performed by standalone EVMScriptExecutor contract. It executes an EVMScript with call data parameters to transfer funds from Aragon Agent using Aragon Finance contract as intermediary.

Enact a motionEasyTrackEVMScriptExecutorAragon FinanceAragon Agent

Considered options

There are several issues needs to be resolved to approach the final design:

1. Where to implement limit checks?
2. Will the limits be checked also at the start of the motion?
3. How to act if spending in different tokens is required within the same budget?

1. Where to implement limit checks

Budgets should to be stored on chain for each Committee. To address the issue we propose to indroduce mechanics similar the Gnosis Allowance module.

In this scenario, DAO assigns ‘allowance’ for the Committee by the on-chain Aragon vote and sets appropriate values for the amount, periods and initial period offset (if needed).

When a Motion is enacted, within the same transaction:

- first a check is started to see if the amount of the Motion is included in the current balance of the budget. Checking if the current payment goes to the budget balance, compares the current budget balance and the payment amount. If the limit is exceeded, the transaction is completely reverted.
- the payment is made.
- the current budget balance should be reduced by the amount of the payment.

To check and enforce the limits, it is necessary to decide where to put checks the financial constraints during the enaction flow?

There are several options for the place where to intercept the calls to check if they are valid within defined budgets and limits.

Option 1. Check limits inside the Aragon Finance contract

Implementation of checking limits inside the Aragon Finance contract has two main Pros:

- it’s upgradable;
- it’s designed to provide budget functionality.

But this solution has too many disadvantages:

1. to check if the amount is within budget, it’s necessary to distinguish exactly which committee (technically, which EVMScriptFactory

) is making the payment. From the Finance point of view Easy Track is the one large entity and it is not designed to distinguish separate factories and have separate budgets;

1. if it is important to call checks not only while executing, but also at the moment of creating a Motion, then the Aragon Finance contract is not appropriate, because it emerges in the end of the enaction flow;
2. if the payment will be made by another contract, not Aragon Finance, then it will be necessary to implement limit checks into it as well;
3. Aragon framework interfaces and agreements are limited in their ability to change. And even assuming that it can be safely implemented, if there is a need to change Aragon to another framework, this solution will not be transportable.

The Aragon Finance option is considered in this ADR rather to understand why this option is the least appropriate. And the main choice will be between options 2 and 3.

Option 2. Check limits inside the EasyTrack contract

EasyTrack contract internal security mechanics can be improved to support limits and budgets. Limit checks can be called inside the EasyTrack contract every time a Motion is created and enacted.

Pros:

- EasyTrack can distinguish EVMScriptFactory

types;

- this will be a common reasonable safety net that will be applied to all the Motions that already exist and will be developed in the future.

Cons:

- It will require redeploying of EasyTrack contract;
- It could induce an abstraction leakage. At the level of EasyTrack there's no data on the amount of payments and other data needed to check limits. Even if the abstraction levels were curated, to obtain this data, a huge redesign of the whole EasyTrack and its redeploy would be required.

Option 3. Check limits inside the EVMScript Factory

Limit checks can be called inside the factories before creating the script.

Right now there are three deployed EVMScriptFactories

that make payments and that need to be capped:

- TopUpLegoProgram (for LEGO program);
- TopUpRewardPrograms (for reWARDs program);
- TopUpRewardPrograms (for Referral program).

Two of them are different instances of the same contract.

Since in this case it's necessary to redeploy all theese factories, it makes sense to create a new general payment factory type for all committees. And deploy an instance of it each time a new committee is created.

The functionality for accounting requests and checking limits will be called inside the factory. At the specification stage, a decision on 'how to implement it' will be made: as a built-in factory's bytecode (extracting a library or a base contract), or as an external oracle-like contract to be leveraged by all of the factories.

Pros:

- easy to implement: inside factories there are already ready data to check limits;
- easy to understand;
- does not require redeploying and reauditing Easy Track itself, only the new code needs to be audited deeply;
- one general payout contract to be used by each committee - looks transparent and secure.

Cons:

- it would require to update all committees and proposals: there many things to communicate with different people and DAO itself;
- nowadays some motion types use only LDO payouts. If they are replaced by generalized-one versions, it might open new undesired behavior and potential unforeseen isssues (It may add another security considerations).

2. Will the limits be checked also at the start of the motion

There are only two time points where limits can be checked or/and enforced: on a motion creation, and on its enactment.

Option 1. Check limits only when a Motion is enacted

The option would require additional actions for each Committee, and limits become opaque from the prospects of UI and external customers.

Option 2. Check limits when a Motion is created and when it's enacted

In this case, when a Motion starts, the check will be called from the UI.

At the start of a Motion, it is not possible to accurately predict when the motion will be enacted. And, for example, in the case of a conversion or in the case of creating a motion in one period with finishing in the next, it is impossible to know exactly what the exact amounts will be to check at the time of the enact. Therefore, at the moment of start, the check will be carried out, and if there is a possibility that the motion may not pass, then a warning will be issued about this. But the user will still have a choice: start the motion anyway or refuse.

If the motion is created on the border of the periods, i.e. starts in one budget period and ends in another, then not the current balance of the budget is taken for verification, but the total budget limit, which is set as the amount to replenish the budget in the new period, and the amount of the motion is compared with it.

If the write-off occurs in tokens other than budget storage tokens, then at the start, the amount of the motion + 20% is taken for verification as a gap for changing the rate.

3. How to act if spending in different tokens is required within the same budget

It is assumed that the amounts in the budget can be in one tokens, and payments in others. In this case, amounts should be converted. There are several options for how to solve this problem:

Option 1. Do not convert at all and forbid transfers in tokens other than in the budget.

This option is not suitable, because there's a need to manage budgets in different tokens.

Option 2. Check budgets by some sort of TWAP.

Looks technically difficult to implement. It can be done off-chain. And there is no assurance that this is correct way to convert amounts.

Option 3. Check budgets by spot exchange rates.

In this case, the limit for the EVMScriptFactory is stored in one specific token. Also, as one of the parameters for each EVMScriptFactory, it is necessary to store a list of tokens in which it is generally possible to write off funds for this factory. At the time of checking whether the amount of a Motion satisfies the limit, if the debit token differs from the limit storage token, then spot exchange rates and Chainlink pricefeeds are used.

Decision Outcome

The following solution is proposed:

Where to implement limit checks

It is proposed go with option 3 and make one generalized EVMScriptFactory for making payments and deploy an instance of it for each of the committees. Limit parameters and limit checks will be implemented as a built-in factory's bytecode or as an external oracle-like contract to be leveraged by all of the factories - that will be decided at the specification stage.

Will the limits be checked also at the start of the motion

The limits will be checked both at the start of the Motion and when it's enacted (option 2). When a Motion starts, the check will be called from the UI. At the moment of start, if there is a possibility that the motion may not pass, then a warning will be issued about this. But the user will still have a choice: start the motion anyway or refuse.

How to act if spending in different tokens is required within the same budget

To solve the issue with payments in different tokens, it is proposed to check budgets by spot exchange rates and use Chainlink pricefeeds to convert amounts for the whitelisted tokens (option 3).