OK, here goes with a technical write-up that hopefully SEO will pick-up

Overview

We’re all lazy and we want to automate menial tasks - think accounting, maximizing staking time in tokens, minimizing transaction fees, paying our bills etc. This is the stuff that’s annoying for us all.

For this write-up, we’ll talk about the perspective of a DAO that collects fees in arbitrary tokens, and wishes to regularly swap these tokens to a target

token.

Objective

Using CoW Protocol architecture we will:

1. Automate fee collection from the DAO’s sub-entities.
2. Swaps fees accumulated into a target

token.

1. Bridges target

token to Ethereum mainnet.

In doing so, we attempt to meet as much of the DAO’s desired experience:

- Keeperless operation - The DAO should not be required to maintain a keeper for the tasks of claiming or swapping tokens.
- Assured swap rates - The DAO should have a mechanism that guarantees favorable swap rates for the accrued fees, eliminating the risk of front-running.
- Frequent fee collection - A system should be in place that enables regular collection of fees.
- Simplified on-boarding of new tokens - The process of adding different tokens that the DAO may have collected should be simplified. Preferably no need to create new contracts or recurrent on-chain voting.
- Easy transition to new target

token - The system should be flexible enough for the DAO to atomically change the target

token to minimize governance overhead.

DAO Technology Stack

For the purposes of this article, this section covers the DAO’s initial

technology stack, prior to automation.

On-Chain

- Agent
- A contract that essentially acts as an “avatar” that voters of the DAO are able to control through proposals / voting to enact proposals. It is assumed that an enacted proposal executes interaction(s) with other contracts as determined by the proposal.
- FeeCollector
- A contract with Agent

as an owner that collects fees. It is assumed the contract has: * setOwner(address)

- set a new owner

.

- withdraw()
- does not

need permission for calling, and withdraws fees accumulated to the owner

’s address.

- setOwner(address)
- set a new owner

.

- withdraw()
- does not

need permission for calling, and withdraws fees accumulated to the owner

’s address.

Off-Chain

It is assumed that there is no current off-chain infrastructure owned / run by the DAO.

Methodology

This section covers the steps that could be used to solve the objectives as best as possible.

On-Chain

New contract instances

Custom Safe

Module: BridgePushModule

Implement a custom Safe

module that exposes permissionlessly a method (such as zap

) that:

1. Checks the xDAI

balance of the Safe

exceeds a minimum threshold

.

1. Bridges xDAI

from Gnosis Chain to Ethereum Mainnet to a target

.

WARNING

: Configuration of the target

in the module should be set to only be modifiable by the Safe

that has the module enabled.

Safe

Deploy a new safe

with the parameters:

- threshold = 1
- owners = [agent]
- fallbackHandler = ExtensibleFallbackHandler(0x4e305935b14627eA57CBDbCfF57e81fd9F240403)
- Modules enabled: BridgePushModule

After the Safe

has been deployed with ExtensibleFallbackHandler

, configure CompoosableCoW

by doing an execTransaction

on the Safe

itself such that:

safe.setDomainVerifier(0xc078f884a2676e1345748b1feace7b0abee5d00ecadb6e574dcdd109a63e8943, 0xF487887DA5a4b4e3eC114FDAd97dc0F785d72738)

NOTE

: The first value above corresponds to the GPv2Settlement.domainSeparator()

for the chain in which the Safe

is being deployed, and subsequently being the same chain that will be used to interact with CoW Protocol!

. The second value corresponds to the ComposableCoW

address for the chain.

At this point, we now have a Safe

that is completely under the control of the Agent

and is capable of multiplexing many different CoW Protocol orders with the use of ComposableCoW

Discussion - Why setup a Safe

and alternatives

This methodology for advanced conditional orders requires

the use of a smart contract and makes use of ERC-1271. As such you have two options:

1. Use an existing smart contract that already supports ERC-1271 isValidSignature(bytes32,bytes)

, and has a good UX for retrieval of tokens / arbitrary transaction support - of which Safe

is perfect for this. Battle-tested and proven.

1. Roll your own ERC-1271

-enabled smart contract and implement the associated token retrieval operations etc in the event funds need to be clawed back from your custom contract.

For those wanting to implement their own ERC-1271

capable contract, ERC1271Forwarder

in the GitHub - cowprotocol/composable-cow: 🐮 Composable Conditional Orders for CoW Protocol repository provides hints as to how to do this.

Extended custom TradeAboveThreshold

conditional order

A TradeAboveThreshold

conditional order type signals an intent to always

sell sellToken

for buyToken

when the balance of sellToken

exceeds a defined amount.

It would be suggested to modify Data

struct, such that:

struct Data { IERC20 buyToken; uint256 threshold; address feeCollector; uint8 coin; }

Discussion - What did we do with this struct

?

With the changes we now can:

1. Infer the sellToken

by querying the feeCollector.coin(i)

for the sellToken

.

1. Infer the current balance

across the feeCollector

and the Safe

by adding coin.balanceOf(safe)

- feeCollector.admin_balances(i)

.

1. If (2) exceeds threshold

, the Safe

signals it’s intent that it wants to trade for buyToken

.

1. We removed target

as we want to ensure that the destination for the Swap

remains the Safe

that was configured.

Off-Chain

Nothing happens automatically on blockchain. There’s no cron

jobs, so something

needs to act as a keeper. This could either be run by a DAO, or the tools made to make it decentralized enough so that someone can trigger it.

New Off-Chain infrastructure

Extend the Web3Actions

for Tenderly to pickup via IPFS

/ GitHub

repo the latest proofs for all the current conditional orders for the active root

(more on the root part below when the complete setup is walked through).

The Web3Actions

MUST

also be adapted such that:

1. If a ConditionalOrder

signals it’s intent to trade, the action adds the pre

hook to instruct the solver to call withdraw

on the FeeCollector

, which will automatically move the collected fees to Safe

for trading. NOTE

: The pre

hook is an API field that is added to the API call to the CoW Protocol API.

1. If after the ConditionalOrder

is executed, it is expected that the target.balanceOf(safe)

exceeds the BridgePushModule.threshold

, the add a post

hook to call BridgePushModule.zap()

.

Discussion - What guarantees are there for pre

and post

hooks?

- pre

hooks MAY

be implicitly

guaranteed when the settle

would otherwise revert

. For example, if the pre

hook called an ERC-2612 permit

method to give an allowance to the GPv2VaultRelayer

without which there is insufficient allowance, it can be said that this is an IMPLICIT

guarantee.

- post

hooks do NOT

have an on-chain guarantee of their success. You MUST

take this into account when designing your mechanisms. The case where this is likely to cause issues is if your mechanism requires post-swap accounting to be complete within the same transaction. In these cases, thought may be given to delaying the accounting until the next

swap transaction where you can guarantee itin a pre

hook (ie. you make the pre

hook require the accounting to be complete before continuing

).

In Production

Now, with all the pieces in place, let's discuss how this would work in production:

1. The DAO makes a proposal to execute from the Agent

:

a. Set all FeeCollector

admins to the Safe

. This can be batched if the Agent

permits delegatecall

by calling via Multisend

.

b. Using the forth-coming cow-sdk

, create a Merkle Root of all ModifiedTradeAboveThreshold

Conditional Orders covering all FeeCollector

and their parameters. Using this root

, the Agent

calls Safe.execTransaction

to do ComposableCoW.setRoot()

. The signature for this execTransaction

can be set to a special signature as it's being called by the only owner (Agent

).

1. Off-chain Infrastructure (such as the Tenderly Web3Action) is running with the root proofs calculated in (1b). This will continually poll each ConditionalOrder

, and therefore check each FeeCollector

whether or not it has accrued above threshold

. If so, it will trade. The added pre

hook will ensure that there are enough funds for the swap in the Safe

(otherwise, the settlement will revert, whose costs are borne by the solver, and so they're incentivised to get it right). The added post

hook will bridge out funds back to mainnet in accordance with our strategy. NOTE

: This transfer back to mainnet doesn't have on-chain guarantees, however the way the system is designed, funds are SAFU and would simply be shuttled to mainnet on the next fee collection.

Review

Now, let’s review how we’ve gone with trying to meet our objectives:

- Keeperless operation
- The DAO still needs to run a keeper, or employ keeper resources. Decentralization of the keeper would be relatively straightforward so that this could be distributed. A novel approach as well could be to add a watchtower functionality into the DAO’s UI that polled a currently viewed FeeCollector

to see if the browser should trigger an order. This may markedly increase RPC

calls on the front-end and could result in CoW Protocol API endpoint spamming.

- Assured swap rates
- In the system outlined thus far, the effective settlement rates depend solely on the CoW Protocol batch auction competition amongst solvers. This would require multiple solvers / highly competitive environments to ensure adequate risk diversification. An alternative would be to modify ModifiedTradeAboveThreshold

to use PriceCheckers

, with the Web3Action

tweaked such that it supplied a nominated rate, with the PriceChecker

verifying this on-chain against an oracle.

- Frequent fee collection
- The proposed system meets this requirement.
- Simplified on-boarding of new tokens
- The proposed system would require a single vote by the DAO per each time adding new tokens. This would require setting an approve

on the Safe

to the GPv2VaultRelayer

for the additional token(s), and setRoot

of the new root of Conditional Orders.

- Easy transition to new target

token

- The proposed system could simply have the buyToken

modified across all the conditional orders. As this is then summarized into a merkle root, it would simply require one vote by the DAO at which the Agent

would trigger the Safe to call

ComposableCoW.setRoot()` with the new root.