CoW Grant – Authenticated Hooks

Author

I am meetmangukiya. I am a software developer, have been in DeFi and web3 space for over 3 years now. I also have familiarity with CoW protocol technicals.

I have also completed the following CoW grants:

1. Automated fee collection - https://forum.cow.fi/t/cow-grant-automated-fee-withdrawal/

Grant Category

Other/Misc.

Grant Description

All hooks, irrespective of the order, are executed from the context of HooksTrampoline

. This is a requirement as otherwise if executed from GPv2Settlement

, this would place accumulated protocol fees in direct jeopardy.

This setup works fine when executing hooks that are permissionless

(such as claim

for an airdrop that's not authenticated on msg.sender

), and also works for hooks that have self-contained authorisation (e.g. permit

, where a user signs the permit, in order to be able to set allowances).

This setup is limited in scope to protocols / standards that already provide some inherent type of signed authentication

, or are completely permissionless

. This prohibits integrations such CDP management, vault savings strategies etc.

Grant goals and impact

This grant introduces a solution to this by standardizing a way to sign the hooks calldata such that it will be authenticated on chain and user-specific user-owned accounts will be called with those hooks data.

It also acts a solution to the problem of post hooks not being atomic. So even if the post hook fails, the assets will still be under the custody of the user-owned proxy, and not lie in the HooksTrampoline contract for anyone to steal.

Scope

To implement a solution that allows for the use of deterministic addresses to be associated with an order owner so that the owner may set the receiver

to an address that hasn't be deployed yet, but which they are guaranteed to have control over.

To this end, implement an EIP-1967 proxy factory. The factory should use CREATE2

to create the proxies with msg.sender

as the salt

. E.g.:

The factory should adhere to the interface:

function executeHooks(Call3[] calldata calls, bytes32 nonce, bytes32 r, bytes32 s, uint8 v);

The type of order owner is determined by reference to v

. If v == 0

, it is assumed to be a smart contract, otherwise an EOA.

EOAs

The hook's owner is determined by ecrecover

of rsv

Smart contracts

The hook's owner is determined by assuming that r

is set to the owner's address. The smart contract is assumed to supportERC-1271.

Deterministic proxy deployment

Within the proxy factory, an internal function is used for deployment:

function _create(address owner) internal { bytes32 salt = bytes32(bytes20(owner)); Proxy proxy = new Proxy{salt: salt} (implementation); }

Execution of hooks

When running executeHooks

in the proxy factory, the proxy factory:

1. Determine's the order's owner
2. With (1), calculates the deterministic address for which their proxy should have been deployed to
3. If (2) has a code length of 0, deploy the deterministic contract, with a delegate call to the implementation

, for trustedExecuteHooks

.

1. Otherwise, call the proxy address with trustedExecuteHooks

.

Implementation

implementation

would be a singleton that adheres to the interface:

interface CoWAuthHook { function executeHooks(Call3[] calldata calls, bytes32 nonce, bytes32 r, bytes32 s, uint8 v); function trustedExecuteHooks(Call3[] calldata calls, bytes32 nonce); function updateTrustedExecutor(address who, bool authorized); }

NOTE

: trustedExecuteHooks

should ensure that msg.sender

is the ProxyFactory as this can then ensure that signature verification has taken place, and is authenticated (no need to do the signature verification again).

State handling

Any state contained within the proxy MUST

be stored using the Diamond Storage methodology to avoid any potential state conflicts on upgrades by a user.

ENS resolver

Provide a forward resolver that will resolve any

subname of cowhooks.eth

(example) to the deterministic proxy address for given

. So, if any user wants to see what is their proxy, they can just search etherscan for

.cowhooks.eth

and get the address they can use as receiver

for their swap orders, etc.

Also, provide a reverse resolution of

.cowhooks.eth

to

. One thing to note is both the forward and reverse resolution won't happen until the said proxy is deployed, since this logic will be encapsulated in the proxy deployment/initialization logic.

Tools

When implementing this solution, the following must be provided:

1. *Smart contracts for the ProxyFactory*

, the implementation

(singleton) and the ENS forward/reverse resolvers.

1. *Comprehensive test suite for the smart contracts*
2. *SDK / Typescript for:*

a. Calculating the deterministic proxy address when deployed via the ProxyFactory

.

b. Signing of the Call3[]

and nonce

EIP-712 type.

c. Encoding of the hooks calldata

Example to be provided

Use a prehook to borrow DAI on behalf of the user and send it to the user. The order will then pull it from the user's EOA during the settlement.

Milestones

1. *Contracts and tests - 2 weeks*
2. *SDK and usage example - 1 weeks*

Funding request

1. *Upfront - 5000 xDAI*
2. *Upon completion - 5000 xDAI*
3. *Upon completion - 10000 COW, vesting over 12 months.*

All of the budget going towards development.

Gnosis Chain Address

gno:0xd33d055ea52ec5a685995d40f19fa98d0744df0c

Terms and Conditions

By submitting this grant application, I acknowledge and agree to be bound by the CoW DAO Participation Agreement and the CoW Grant Terms and Conditions.