Gasless Miner-Subsidized Transactions

Authors

@vicnaum & @marcellobardus (Oiler Network)

Thanks to Tomasz Stanczak (Nethermind) for collaboration, comments and review.

Abstract

Here we describe a method of gasless transactions via Flashbots and MEV, without the need of account abstraction, GSN compatibility or whatsoever - making the user's payload transactions completely standard and potentially submittable via standard wallets (Metamask, etc - with some changes to the UI).

History

Searcher sponsored transactions were known for a while: searcher sponsored tx | Flashbots Docs

These were generally used just to make calls from an account that has a compromised private key.

We try to go further and allow any account to be subsidized by miner, with a return of subsidy in the end of the bundle with a MEV fee as a premium for this service.

Preface (Basic Example)

Say there is a transaction in the mempool from Wallet A which says "Send 10 000 USDC to Wallet B"

.

But there is a problem: Wallet A [User] doesn't have any ETH on its balance to pay for gas - so the transaction is in the limbo and cannot be included on the chain, and currently it will be removed from mempool in a short time.

Wallet B [Miner] notices this transaction in the mempool and (acting as a MEV searcher) sends some ETH (required for gas payment) to Wallet A - resulting that now Wallet A has ETH balance and the transaction will succeed and can be included.

This results in 10 000 USDC being transfered to Wallet B - so receives compensation for ETH send plus profit.

Rationale

The above scenario can be generalized and extended from a single subsidized transaction to a bundle of Payload transactions with a Return transaction in the end:

- TX1 [SubsidyTX]
- MEV Searcher sends some ETH to Wallet A
- TX2 [PayloadTXs]
- Can be either a single TX or a Bundle that does any useful action for Wallet A (Uniswap swap, or whatever)
- TX3 [ReturnTX]
- Wallet A returns the favor to Miner (block.coinbase). This can be paid in any currency (ETH, USDC, etc) and includes the subsidy from TX1 + premium incentive fee (MEV).

This construction is completely safe for everyone:

- TX3 cannot be executed before TX2, because it has a higher nonce - so nobody can't just steal the money without first executing TX2.
- TX2 cannot be executed until Wallet A has received ETH on balance to pay for the gas.
- TX1 is included in the block by Miner only if they are sure that TX2+TX3 bundle will not revert after providing ETH for gas with TX1 and if TX3 refunds enough funds + desired premium.

TX2+TX3 can be submitted by Wallet A as a Bundle (Flashbots?).

And the Bundle can be verified to meet the above conditions and included into the block when MEV from it is extractable.

The Bundle can have many transactions - approvals, swaps, etc - and as their nonces are sequential - the same rule applies: they all have to be executed for the last ReturnTX to succeed and refund the Miner.

Reorganization risk

If a Miner finds such a bundle and broadcasts his TX1 with it - there is always a small chance of network reorganization happening. If it takes place and the block gets uncled - then the miner TX1 will still send money to Wallet A (cause signed transaction for it is already public), but TX3 will refund to a new miner, who did the reorganization.

To mitigate this issue - we propose to use a smart-contract that protects TX1 by checking if block.coinbase

(current block miner) is equal to a TX1 msg.sender

- otherwise the TX1 transaction will revert.

Implementation details

Implementation requires a simple contract to be deployed with two functions for TX1 and TX3.

TX1 [SubsidyTX]

protects Miner from Reorgs/Uncles (checking that the Miner of the block is the same address that subsidizes User):

function subsidizeGas(address payable receiver, uint256 amount) public payable { require(msg.sender == block.coinbase, "You're not a miner of the block"); receiver.transfer(amount); }

TX3 [RefundTX]

should be submitted by Wallet A in the end of the bundle to refund the Miner of the block:

function refundToMiner(address token, uint256 amount) public payable { if (token == address(0)) { block.coinbase.transfer(amount); } else { IERC20(token).transferFrom(msg.sender, block.coinbase, amount); } }

Possible issues

1. Cancellation of Payload transactions of the User: if a user changed their mind or wants to add Ether themselves. But if this bundle comes through Flashbots - cancellation should be handled via Flashbots API.

Variations

1. A simplified version can be executed, where instead of TX3 [RefundTX]

User can just submit transactions with a high PriorityFee - which is a little bit higher than the baseFee to include Premium - cause PriorityFee also is basically a refund to Miner. Pros of this variation: no TX3 is needed, no contract interaction is needed for User. Cons: User can only refund in ETH this way.

1. A more generalized system can be built where not only Miners can subsidize, but anyone, through the pooled smart-contract, and refunds will go back to the smart-contract with a premium, earning the whole pool some yield. The only problem here is reorganizations - if a reorg happens - miner might not want to include the final Refund TX3, and the pool will lose the subsidy funds. But with POS (assuming it can't have reorgs) - this kind of system might work.

Example (on Mainnet)

To put money where our mouth is, we created an incentive:

There is signed transaction from wallet 0x307505150d6b28E89e57C4699959957Cb4964d26

on Ethereum Mainnet that will send 100 USDC to a Miner:

0x02f8b10103843b9aca008523284d26008301d4c094620296909a4cc9ff61a20aa13fd83fb822f4dc5c80b8440737d414000000000000000000000000a0b86991c6218b36c1d19d4a2e9eb0ce3606eb4800

Wallet doesn’t have any ETH balance (only dust).

To unlock this transaction - Miner should send 0.01812 ETH to the above wallet.

This can be done via contract: 0x620296909a4cc9fF61A20Aa13FD83fb822f4Dc5c

by calling subsidizeGas()