barrywhitehat, Lakshman Sankar

Introduction

For private applications on block chain we need someone to pay for the gas that will eventually be refunded.

We need a way for relayers to advertize their willingess to pay for a transaction. This needs to be resistant to spam.

So a user should be able to select relayers and have high confidence that they will actually broadcast a transaction.

Desiried properties

1. The target contract does not need to know about the relayer registry contract. So we can switch it out in the UI and don't need any contract change.
2. The advertisment mechanism is spam resistant.
3. Users have high confidence that their transaction will eventually get mined.

Mechanism

Every transaction that the relayer processes passes through the relayer registry contract. This contract sends the broadcast request to the target contract which sends the fee to the relayer and the remaining funds somewhere else.

The registry contract then burns a set % of the fee and sends the rest back to the relayer. The registry contract tracks the amount of funds that has been burned by each relayer.

The burn acts as an economic cost that every relayer needs to pay in order to relay transactions.

To overcome the spam problem we require that a min_percent

% of all funds that pass through the registry are burned.

When we look at the list of relayers ordered by burn amount we have a list of actors that either

1. Burned a large amount of their own money in order to get high on that list

or

1. Processed a lot of transactions from other users.

If they are no 1. that means they have spent a bunch of capital and are now trying to make it back.

If they are no 2. That means that their past behavior has caused them to act as reliable relayers.

Relayer infrastructure

Each relayer lists an ip addresses (or tor address or domain names) in the relayer registry contract. Users select a relayer from the registry, presumably using the quantity burned by each relayer as a proxy for reliability. We may implement a UI that randomly selects a relayer weighted by quantity burned.

The transaction must be well formed as a legal ethereum transaction but the account that signed that transaction can have 0 balance.

The relayer takes the transaction, changes msg.sender, resigns that transaction and executes it in the local EVM. If the transaction results in their balance being increased they broadcast the transaction to the network.

Front running

Each relayer will have their own break even point and will not broadcast transactions that pay a fee less than this mark.

A relayer with a very low break even point can monitor the tx pool and can rebroadcast transactions with higher gasPrice, thus reducing the time for transactions to mine. And stealing transactions from relayers who use a high gasPrice.

This will result in a market place where people are competing to broadcast a transaction.

Miner overtake

Taking this idea to its logical conclusion miners would be able to front run transactions that the relayers broadcast. But this is the status quo of ethereum where miners can decide the ordering of transactions and get paid the fee. If we can get miners to accept transactions sent via the relayer registry contract then we have achieved the goal of transaction abstraction.

Paying in differnt tokens

Relayers can accept any token they wish and the relayer will accept erc20 tokens or eth. Each token has the gas burn. Users slect a relayer based upon a burn in a given token.

Using this with arbitrary contracts

We can use this with any contract we like by simply deploying a contract that sends the refund for gas to an address that is defined by the user.

This means that our users can upgrade to another transaction abstraction method. Its also possible for them to bypass the trasaction abstraction altogher.

This puts the users firmly in ocntrol and avoids a tight coupling between the mixer and the relayer registry contract.

DoS attacks

It is possible for a user to DoS a relayer. For example a user could

1. Ask a relayer to relay a transacation and then relay it themselves which a higher gas price
2. Craft a smart contract that appears as if the relayer will get funds when they test with their local node. But fails on the main net. Somthing like only working with even blockNumbers

would work 50% of the time.

Number 1. is a greifing attack where the gas is the cost to the user and the gas of the relayer is the amount that the victim loses. So unless the attacker is a miner the attacker still has to pay a fee and likly more than the victim.

The victim can also monitor the tx pool for transactions that would supersede their transaction or invalidate it and rebroadcast a new transaction with the same nonce and a slightly higher gas fee if they notice they are being attacked.

Case 2. can be alleviated by using a different fee for recipient contracts. For example I could require a really high fee for a brand new contract and slowly reduce it over time. Relayers can also use the amount of eth burned as a way to set their own fee. If a large amount of eth has been burned by a contract that means its probably not specifically desiged to greif relayers.

Both of these kinds of attacks are currently deployed against front runners who still persist my increasing their minFee they are willing to try and get. So its likely that both can be absorbed by some relayers as cost of doing business.

Conclusion

Burning a small % of fees lets us have a spam resistant list of reliable relayers who either have

1. Burned an amount of their own funds
2. Broadcast a large number of transactions.

The more the system is used the more funds someone needs to burn in order to get to the top of the list.

If a relayer gets to the top of the list and then starts to charge higher fees the front running mechanism will allow other broadcasters to take their fees and increase their own burn eventually overtaking them in the list.