Proof of Concept for Copy Funds on @melonproject/protocol

Table of Contents

Introduction and Preparation

This documentation aims to explain how we implemented the PoC-stage of copyFunds-module for the @melonproject/protocol. In general this module enables fundmanagers to copy a whole fund with a given investment amount (`copyFunds(fundaddress, investmentAmount)`).

To generate a Proof of Concept setup we need to be able to:

1. Get the holdings of a given fund
2. Compute the shares of every asset from the fund
3. Generate the buy-values of every asset using the total investment amount and the computed shares from step 2
4. Make the Orders with the values from step 3
5. Sell the copied Fund (all assets from step 4)

Since we are using a Testnet and not the Ethereum Mainnet, we dont have a high liquidity on the exchanges and furthermore there are some restrictions which made it really complicated to implement a PoC test-setup. We explain that more detailed at step 4. and 5.

Overall this documentation is about `copyFund.js` which realizes step 1-4 and `sellCopiedFund.js` which realizes step 5. All needed functions for those two scripts are mostly implemented in `wrapperMelon.js`. Also important is the `copyFundsLogger.sol` contract, which is deployed also on Kovan and is used to be able to sell the copied funds again.

The `poc.js` script uses all needed functions and is thought to be the "test-case" for the Proof of Concept.

Explanations

Here we give a detailed overview of the essential functions and also provide explanations of them.

1. Get holdings

The very first and simplest step is to get the holdings of "logged in" (metamask, ec.) account and of a given address. Both are implemented in `wrapperMelon.js`.

```
47  var getHoldings = async () => {
48    try {
49      var manager = await getManager();
50      var accountingAddress = (await Protocol.managersToRoutes(manager, manager.deployment.melonContracts.version, manager.wallet.address)).accounting
51      var holdings = await Protocol.getFundHoldings(manager, accountingAddress);
52      return holdings;
53    } catch (e) {
54      console.log('Could not load holdings: ' + e.message)
55    }
56  }
```

```
69  var getHoldingsOf = async (fundManagerAddress) => {
70    try {
71      var manager = await getManagerOf(fundManagerAddress);
72      var accountingAddress = (await Protocol.managersToRoutes(manager, manager.deployment.melonContracts.version, manager.wallet.address)).accounting
73      var holdings = await Protocol.getFundHoldings(manager, accountingAddress);
74      return holdings;
75    } catch (e) {
76      console.log('Could not load holdings: ' + e.message)
77    }
78  }
```

Computations for 2. and 3.

To be able to copy a fund we need to know the shares of every asset, so that we then can multiply our investment amount with every share and we get the amount we need to buy of this given asset. These calculations are made in `copyFund.js` after getting the holdings.

```
31      const aum = await calculateAUMwithoutWETH(holdingsOf)
32      var assets = []
33      var values = []
34      for (var holding in holdingsOf) {
35        //(WETH and) MLN arent tradeable on oasisdex for now
36        if (holdings[holding].token.symbol != 'WETH' &&
37          holdings[holding].token.symbol != 'MLN') {
38          //get Numbers
39          var aumOf = await calculateAUMof(holdings[holding]);
40          var rate = await getRate(holdings[holding]);
41
42          //do calculations
43          var valueWETH = (aumOf / aum) * _investAmount;
44          var valueToken = valueWETH / rate ;
```

As you may see, first of all we compute the aum of all holdings without WETH, since WETH is almost everytime the basecurrency to trade. To get the share of an asset we just need to get the aum of this asset and divide it by aum of all assets together. Multiplied by investmentAmount we get the value in WETH we need to buy of this asset. To get the amount of token we just need to divide the `valueWETH` by the WETH-rate for the token.

4. Make orders

To place the orders on exchanges we use `makeOasisDexOrder` delivered by @melonproject/protocol and wrapped it for our use to `makeOrder` ind `wrapperMelon.js`.

```
46          //Check if the _investAmount of the given tradingPair is big enough to suceed.
47          if (valueWETH > Math.pow(10,-15)) {
48            console.log(await makeOrder(
49              holdings[holding].token.symbol,
50              valueWETH,
51              valueToken,
52              'BUY')
53            )
54            assets.push(holdings[holding].token.address)
55            values.push(holdings[holding].quantity / Math.pow(1, holdings[holding].token.decimals))
56            //TimeCatcher till order executed. TODO
57          }
58        }
59      }
60      //log in smartContract
61      await logFund(_fundAddress, _investAmount, assets, values)
```

To be able to sell a copied fund we created a logging contract, `copyFundsLogger.sol`. On the bottom of the picture you see the function which logs the needed infos into the contract on the Blockchain.

5. Sell copied fund

The logger makes this very simple. We get the logged fund by `getLoggedFund(fundaddress)` implemented also in `wrapperMelon.js`. To get the specific amount of WETH we just multiply the token amount we had bought by the current WETH-rate. Like the buyorders we use `makeOasisDexOrder` to open the sellorders.

```
 8  var sellCopiedFund = async (_fundAddress) => {
 9      var manager = await getManager()
10      var prepare = await getLoggedFund(_fundAddress)
11      var fund = {
12        manager: prepare[0],
13        amount: prepare[1],
14        assets: prepare[2],
15        values: prepare[3]
16      }
17      for (var asset in fund.assets) {
18          //var tokenContract = eth.contract(tokenABI).at(tokenAddress);
19          var holding = {token: await getTokenByAddress(manager, fund.assets[asset])}
20          var rate = await getRate(holding)
21          var valueWETH = rate * (fund.values[asset] / Math.pow(10,18))
22          var valueToken = (fund.values[asset] / Math.pow(10,18))
23          console.log(await makeOrder(
24              holding.token.symbol,
25              valueWETH,
26              valueToken,
27              'SELL')
28          )
29          //TimeCatcher till order executed. TODO
30      }
31      //unlog in smartContract
32      await unlogFund(_fundAddress)
```

Also we need to unlog the sold fund in the logger contract on the Blockchain. This is done via `unlogFund(fundaddress)`.

Testing

copyFund()

Since opening several makeOrders is not possible, because one can only have one open makeOrder, we tried to copy just two assets, ZRX and BAT from destFund to the srcFund. Therefore we need to make a buy makeOrder from srcFund which then can be taken by by the dest fund. Here you see the described implementations from `poc.js`:

```
                order = await makeOrderPoC(
                    process.env.PRIVATE_KEYsrc,
                    holdingsOf[holding].token.symbol,
                    valueWETH,
                    valueToken,
                    'BUY')
                console.log(order)
                assets.push(holdingsOf[holding].token.address)
                values.push(holdingsOf[holding].quantity / Math.pow(1, holdingsOf[holding].token.decimals))
                investAmount += valueWETH
                //PoC takeOrder by destFund
                console.log(await takeOrderPoC(
                    process.env.PRIVATE_KEYdest,
                    order.id)
                )
            }
        }
        //log in smartContract
        var fundAddress = await getManagerPoC(process.env.PRIVATE_KEYdest).wallet.address
        var log = await logFund(fundAddress, investAmount, assets, values)
```

We should have seen two successfully bought new assets at srcFund, see it logged in the loggingContract, and then sell those two assets again by using the loggingContract. In the end the loggingContract needs to be empty for `getLoggedFund` by the srcFund.

The first asset went through as aimed, but then the second makeOrder failed, because the exchange still thought srcFund had an open makeOrder. That was not the case since destFund directly executed `takeOrderPoC` from `wrapperMelon.js` successfully and also getHoldings showed the asset was transfered (see appendix). We expect this issue on thirdParty side, i.e. the exchange.

sellCopiedFund()

Since the `copyFund()` failed we needed to manually log a copied fund. In contrast to buyorders, one can have several open sellorders. So using a manually logged fund for `sellCopiedFund()` worked as expected, opening several sellorders.

Conclusion

As mentioned before, our analysis of the occurred errors came to the conclusion, that the problem is not on our side, rather than at the exchanges or even the protocol itself. Until further information we see the module "copy funds" successfully implemented at the Proof of Concept stage.

Appendix

Holdings before testing

1.1 srcFund Holdings before all testing:

```
[ { token:
     { address: '0xd0A1E359811322d97991E03f863a0C30C2cF029C',
       decimals: 18,
       symbol: 'WETH' },
    quantity: { [String: '296432170017164412'] value: [JSBI] } },
  { token:
     { address: '0x2C2edf394638931eb672BD9261d2AA1934874d45',
       decimals: 18,
       symbol: 'MLN' },
    quantity: { [String: '34648053943224245570'] value: [JSBI] } },
  { token:
     { address: '0x0A3610a0E87cEDDEE6b81b62b462c7a0fD450E2a',
       decimals: 18,
       symbol: 'ZRX' },
    quantity: { [String: '5997594648800797700'] value: [JSBI] } },
  { token:
     { address: '0xB5098BAFbF90F278374EcFA973A703fD0eb87A12',
       decimals: 18,
       symbol: 'KNC' },
    quantity: { [String: '1332433100000000000'] value: [JSBI] } },
  { token:
     { address: '0xB14c0f4a8150c028806bE46Afb5214daea870CB7',
       decimals: 18,
       symbol: 'BAT' },
    quantity: { [String: '826000'] value: [JSBI] } } ]
```

1.2 destFund Holdings before all testings:

```
[ { token:
     { address: '0xd0A1E359811322d97991E03f863a0C30C2cF029C',
       decimals: 18,
       symbol: 'WETH' },
    quantity: { [String: '8214845957115931437'] value: [JSBI] } },
  { token:
     { address: '0xbdaD7a926A7E70C6B0AF367d97D992b904BBAFcf',
       decimals: 18,
       symbol: 'MKR' },
    quantity: { [String: '2142617280802147757'] value: [JSBI] } },
  { token:
     { address: '0xB14c0f4a8150c028806bE46Afb5214daea870CB7',
       decimals: 18,
       symbol: 'BAT' },
    quantity: { [String: '1672298522086636995000'] value: [JSBI] } },
  { token:
     { address: '0x2C2edf394638931eb672BD9261d2AA1934874d45',
       decimals: 18,
       symbol: 'MLN' },
    quantity: { [String: '11205531574828549980'] value: [JSBI] } },
  { token:
     { address: '0x0A3610a0E87cEDDEE6b81b62b462c7a0fD450E2a',
       decimals: 18,
       symbol: 'ZRX' },
    quantity: { [String: '1058953012014463346000'] value: [JSBI] } },
  { token:
     { address: '0x1D3bC44DD6C3F00640A6825B48F1C78770fd21d8',
       decimals: 18,
       symbol: 'DAI' },
    quantity: { [String: '6573099214581025700'] value: [JSBI] } },
  { token:
     { address: '0xa80C98433E2a82DF3636ED934083E3285163Fad8',
       decimals: 18,
       symbol: 'REP' },
    quantity: { [String: '2471212735553859620'] value: [JSBI] } } ]
```

Testinglogs

First successfull bought asset, error at the second tx:

```
Loaded srcManager: 0x88D855BdF87b93B956154714109d9a5A22A6AD9B
Loaded destManager: 0xB9820Ab5aB6256003124cecE3aFE8140F7e55E15
#####################COPY FUND#################################
0.01017843718592349
{ buy:
   { token:
      { address: '0xB14c0f4a8150c028806bE46Afb5214daea870CB7',
        decimals: 18,
        symbol: 'BAT' },
     quantity: { [String: '1000000000000000000'] value: [JSBI] } },
  id: 37494,
  maker: '0x34B55262cF8367E4c799Bf3008F05fF0070b918c',
  matched: false,
  sell:
   { token:
      { address: '0xd0A1E359811322d97991E03f863a0C30C2cF029C',
        decimals: 18,
        symbol: 'WETH' },
     quantity: { [String: '10178437185923490'] value: [JSBI] } },
  timestamp: '1584451952' }
{ buy:
   { token:
      { address: '0xB14c0f4a8150c028806bE46Afb5214daea870CB7',
        decimals: 18,
        symbol: 'BAT' },
     quantity: { [String: '10178437185923490'] value: [JSBI] } },
  id: 37494,
  maker: [String: '0x34B55262cF8367E4c799Bf3008F05fF0070b918c'],
  sell:
   { token:
      { address: '0xd0A1E359811322d97991E03f863a0C30C2cF029C',
        decimals: 18,
        symbol: 'WETH' },
     quantity: { [String: '1000000000000000000'] value: [JSBI] } },
  taker: [String: '0x0A0DEB797f9138FC93DC020a095f1C9f8d92B690'],
  timestamp: '1584451960' }
0.01234692257487108
Could not makeOrderPoC There is already an open order with token WETH
undefined
TypeError: Cannot read property 'id' of undefined
    at copyFundPoC (/home/ubugo/github/midas/tester/PoC/poc.js:63:23)
    at process._tickCallback (internal/process/next_tick.js:68:7)
undefined
######################GET LOGGED FUND##############################
false
```

changed holdings

srcFund:

```
  { token:
     { address: '0xB14c0f4a8150c028806bE46Afb5214daea870CB7',
       decimals: 18,
       symbol: 'BAT' },
    quantity: { [String: '10000000000000826000'] value: [JSBI] } }
```

destFund:

```
  { token:
     { address: '0xB14c0f4a8150c028806bE46Afb5214daea870CB7',
       decimals: 18,
       symbol: 'BAT' },
    quantity: { [String: '1662298522086636995000'] value: [JSBI] } }
```

Error explanations

We see that this order was traded successfull, but we got the error for the second tx. This could be an exchangeFreeze for several minutes i.e.

Output two trade the ZRX token gives this:

```
Loaded srcManager: 0x88D855BdF87b93B956154714109d9a5A22A6AD9B
Loaded destManager: 0xB9820Ab5aB6256003124cecE3aFE8140F7e55E15
######################COPY FUND#################################
0.012420416161626261
Could not makeOrderPoC There is already an open order with token WETH
undefined
TypeError: Cannot read property 'id' of undefined
    at copyFundPoC (/home/ubugo/github/midas/tester/PoC/poc.js:63:23)
    at process._tickCallback (internal/process/next_tick.js:68:7)
undefined
```

and checking after 15 minutes again is still giving this error. But it should not be there, because the order was directly taken and even `getOrders()` shows not the `makeOrder id 37494`.

Trying to sell it individually gives this weird error:

```
Could not makeOrder Insufficient BAT. Got: 0.000000, need: 10.000000
undefined
```

But checking via getHoldings the returned value is `10.000000000000826`