givethModule

Table of Contents

This is the documentation of the giveth Module from Midas Technologies AG for the @melonproject/protocol.

Testing

IMPORTANT	For now, all tests below are on the **ropsten testnet**.

1. Stage

The first stage is just about the Bridge Contract from giveth (@giveht/giveth-bridge).

TODO'S:

- ☑ Deploy the Bridge.
- ☑ Donate ETH.
- ☑ Donate ERC20 token.

NOTE	We use the WETH token `0x758E94c97caf81d0d0624B272278fe9cd2bdDfB8` and we modified the `giveth-bridge.sol` for test purposes. Especially we added to events and some Errormessages for the `require()` function. Nothing that changes the functionality!

1. Deploy the giveth-bridge.sol

Constructor Arguments are:

- absoluteMinTimeLock: `uint256 90,000`
- escapeHatchCaller: `address 0x812ea1c4C193Ffa12a3789405E3050a066FCbE25`
- escapeHatchDestination: `address 0x812ea1c4C193Ffa12a3789405E3050a066FCbE25`
- maxSecurityGuardDelay: `uint256 432,000`
- securityGuard: `address 0x812ea1c4C193Ffa12a3789405E3050a066FCbE25`
- timeLock: `uint256 172,800`

The deployed address is `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655`.

Observables:

- gas Used: `1,931,528`

Now we are able to test the functions of the Bridge we need.

2. Test donate Ether on the Bridge

The following steps are required to donate ETH to a DAC (here just as example `receiverID = 1`).

1. Execute `donateAndCreateGiver(address giver, uint64 receiver)` from `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e` with the inputs:
    - `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`
    - `1`
2. Tx hash: `0x72d4d506275409fb5010fc12719f41c4eee28a06ed38fe5923e7501f6c875f39`.
3. Observables:
    - Gas used: `29,738`
    - Events:
        - Testing:
            - messageSender from called function: `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`
            - messageSender from internal called function: `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`
            - seeThis from internal called function: `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655`
        - Origin:
            - DonateAndCreateGiver

Everything worked as expected.

3. Test donate ERC20 on the Bridge

IMPORTANT	This works without a whitelisting on the Bridge, because we added the constructor to whitelist our WETH token.

First testrun will be without an ERC20 approval for the Bridge. Expected behaiviour is to get a failing tx at the internal tx from the WETH contract.

1. Execute `donateAndCreateGiver(address giver, uint64 receiver, address token, uint amount)` from `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e` with the inputs:
   - `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`
   - `1`
   - `0x758E94c97caf81d0d0624B272278fe9cd2bdDfB8`
   - `1000000000000000` which is 0.001 WETH.
2. Successfull not successfull :) (Tx Hash: 0x64b0eda6983ba481b43d11cf8d532...)
3. Observables:
   - Gas used: `32,905`
   - InternalTx:
     - 1 from `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655` (the Bridge) to `0x758E94c97caf81d0d0624B272278fe9cd2bdDfB8` (WETH token).

NOTE	There are no events triggered.

WARNING	Actually we expect another Error message than `Fail`, because the following picture shows the message we want from the tokentransfer if it fails...

```
892             require(ERC20(token).transferFrom(msg.sender, this, amount),
893                 "ERC20(token).transferFrom(msg.sender, this, amount) failed"
894             );
```

Now we approve `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655` (the Bridge) for `1000000000000000` (0.001).

Execute `approve(address guy, uint amout)` on the WETH token contract from `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`. (See here.)

Since we approved the Bridge Contract we can **repeat 1.** from above:

1. Execute `donateAndCreateGiver(⋯` with the same inputs.
2. Worked as expected.
3. Observables:
   - Gas used: `54,740`
   - InternalTX:
     - 1 from `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655`(the Bridge) to `0x758E94c97caf81d0d0624B272278fe9cd2bdDfB8`(WETH token).
   - Events:
     - Testing:
       - messageSender from called function: `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`
       - messageSender from internal called function: `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e`
       - seeThis from internal called function: `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655`
     - Origin:
       - Transfer from `0x173add8c7e4f7034e9ca41c5d2d8a0a986fd427e` amount `1000000000000000` to `0xA2C8a0F4Fa277c9c50f5B36C7DDC5C86404dA655` (ERC20 Event)
       - DonateAndCreateGiver (Bridge Contract Event)

Here you see the WETH held by the Bridge…

This worked smooth. Now we need to summarize.

Summary 1. Stage testing

We have finished the 1. Stage testing successfully. A few things we need to save for further testing.

1. To transfer the ERC20 token we need to set an approval on the ERC20 token contract. (In this stage we needed to approve the Bridge.)
2. The gas used for an ERC20 donation directly via the Bridge was around `55000`.
3. To donate ETH directly the contract needed around `30000` gas.

2. Stage

The second stage is about the new giveth module in the fork from @midas-technologies-ag/protocol.

TODO'S:

- [x] Deploy the Giveth contract with the previous deployed Bridge.
- [x] Donate ETH.
- [ ] Donate ERC20 token.

NOTE	Again we use the WETH token `0x758E94c97caf81d0d0624B272278fe9cd2bdDfB8` and furthermore previous tests always failed for donate an ERC20 token. Suggestion is that the call stack is too deep.

To visualize the call stack see the following tx-chain:

BaseAddress tx to → 1. execute `donateAsset(⋯)` **Giveth function**, which results in tx to → 2. execute `donateAndCreateGiver(⋯)` **Bridge function**, which results in tx to → 3. execute `transferFrom(⋯)` **ERC20 function**.

In `Giveth.sol` as well as in `giveth-bridge.sol` are the events **messageSender** implemented. This is usefull to track, where we need to set the approval and where the failures are located.

1. deploy Giveth.sol

Expect fine behaiviour with no errors.

IMPORTANT	Set the new Bridge in the constructor for Giveth.sol

Successfull deployed.

Observables:

```
gas Used: `500,920`
```

2. Test donate Ether on Giveth.sol

Expect a successfull transaction which donates Ether to the Giveth-Bridge and creates one internal tx.

BaseAddress tx to→ execute `donateETH()` on Giveth,which calls (and send the ETH of baseTX) via an internalTX to→ execute `donateAndCreateGiver(address,receiver)` on GivethBridge.

Expected Events:

1. on Giveth:

```
Just one messageSend showing baseAddress.
```

2. on Bridge:

```
2 x messageSend showing givethAddress and 1 x seeThis showing bridgeAddress
```

1. `donateETH()` with send Value `0.0005 ETH` Was not successfull, because the provided gas was too less (`50,000`).
2. Success :)
3. Observables:
   - gas used: `62,844`
   - InternalTX: 1 x call from Giveth to Bridge (See here.)
   - Events are matching the expected + the Origin ones. Check Events.

3. Test donate ERC20 on the Bridge

The stack is the deepest on this test.

BaseAddress tx to→ execute `donateAsset(⋯)` on Giveth,which calls via an internalTX to→ execute `donateAndCreateGiver(address,receiver,address,uint)` on GivethBridge, which calls an internal function, which then calls via an internalTX to→ execute `transferFrom(msg.sender,this,amount)` on the ERC20 contract of the token.

Expected new implemented Events:

1. on Giveth:

```
Just one messageSend showing baseAddress.
```

2. on Bridge:

```
2 x messageSend showing givethAddress and 1 x seeThis showing bridgeAddress
```

NOTE	If the donation is successfull we expect the following origin Events: 1. on Giveth,donated(...) 2. on Bridge, DonateAndCreateGiver(...) and 3. on ERC20 Token, Transfer(...)

First tx without approval expected to fail. Since donating ETH via the Giveth to the Bridge was about twice as directly donating at the Bridge and the ERC20 transfer tx needs also gas, we provide `250,000` gas for now.

1. `donateAsset(WETH,1000000000000000)` from baseAddress.
2. `Fail with error: Donations was not successfull`
3. Observables:
   - gas used: `26,415`
   - InternalTX: call from Giveth to Bridge

NOTE	no events since it failed at the very early stage.

New try with more Gas, i.e. `500,000` results in exactly the same...(See here.)

See here with 5,000,000 gas.

Now lets approve the Bridge. Done.

Still the same error.

Now lets approve also Giveth. Done.

Still the same error.

Now we changed `abi.encodeWithSignature()` to `abi.encodePacked()`.

deployed.

But also fails with nearly the same obsevables. (See here.)

Now with `bytes8(keccak256(⋯))` here.

Also fails…

Analisys

TODO!

possible ways:

- transfer the WETH to bridge or giveth
- try other abi.encode…

Summary

TODO’S

- ▢ Testing 2. Stage

Maybe:

- ▢ Try error msgs on Bridge via direct call.