MultiversX Paymaster

MultiversX smart contract - Security audit by Arda

Repository: *https://github.com/multiversx/mx-contracts-rs*
Smart contract path: *contracts/paymaster*
Initial commit: *428c8dbe2e2f4296fb8c3bff90678d5ac28f9b39*
Final commit: *d857fedafa2a18ad59eca85ac3ba7cfb9d77dee1*

Issues

A.1. Solved Critical **If the forwarded call fails, the user's funds are lost.** This is because in the callback `transfer_callback`, we read the back transfers in order to catch all payments and send them back to the user, however for failed asynchronous calls back transfers are empty. Rather, the initial payments which are returned to the Paymaster SC are present in the call inputs.

Solution: In `transfer_callback`, in order to determine the payments to send back to the user, we suggest:

- If the asynchronous call succeeded, we get payments from back transfers.
- If the asynchronous call failed, we get payments from arguments given to the callback function.

We further suggest adding a unit test to verify that the user correctly receives his payment when the transaction has failed.

A.2. Solved Major BackTransfers do not fully work with cross-shard async call v2. So in case of a cross-shard call, the user might lose tokens as they would not be caught in the callback and not sent back to the user.

Moreover, tokens stuck in the contract can further be drained by any user whose tx to the Paymaster performs a built-in function call.

Solution: We recommend checking that the callee is in the same shard. There would be 1 paymaster SC deployed per shard.

A.3. Solved Major User can't protect against too small gas amounts provided by relayer. Namely, a relayer could provide a too small amount of gas for the async call to succeed. The call would fail, and the relayer would have taken the fee payment.

Solution: We recommend adding a gas limit argument to the endpoint `forward_execution`, that the user can set to protect himself. We would require in the endpoint that the gas left is greater than this limit.

A.4. Solved Medium Transfers back from SC on remote shards will fail if SC is not payable. Indeed, for cross-shard calls, only the last payment from callee to caller is authorized, while the others are forbidden (unless the caller SC is payable).

Solution: We recommend having 1 Paymaster SC per shard, and enforce that calls are made to SC in same shard (as in A.1.)

A.5. Solved Medium The Paymaster SC does not send any token back to the user in case of a successful transaction. So for example in the context of a gas-less swap, if we call xExchange pool to make a swap, the Paymaster SC does not send the output back to the user. This is because right now, the Paymaster only executes the SC endpoint and sends nothing back to the user.

Solution: For the purpose of gas-less swap, we suggest introducing a new SC which takes care of making the swap (and other operations if needed) and sending the tokens to the user. The Paymaster SC would then call that endpoint.

A.6. Solved Minor There is an obsolete //TODO comment in `transfer_callback`.

Solution: We recommend removing it.

A.7. Solved Minor There is no test for the case where the async call of `forward_call` fails, to check that the user gets back his payments as expected.

A.8. Solved Minor Misleading comment can be deleted:

```
/// An empty contract. To be used as a template when starting a new
contract from scratch.
```

A.9. Solved Minor Unnecessary `sc_print` in the callback `transfer_callback`.

A.10. Not Solved Minor A safety check in `forward_execution` that the payment length is smaller than a constant `MAX_PAYMENTS_LENGTH` would prevent some edge cases from happening if the number of payments is too big:

- Could make `forward_call` run out of gas when calling remove, and in this case the relayer loses his gas fees.
- Could make `transfer_callback` run out of gas when trying to send all payments back to the user, and in this case the user has lost funds.

A.11. Not Solved Minor The const `ESDT_TRANSFER_FUNC_NAME` is useless (at review commit *18a6ad3cf8cdcf30e9ab86e13356174105ab04a1*).

Inherent Risks

1. A user pays the relayer fee even if their smart contract call fails.

Disclaimer