Audit report

AdEx Protocol

Repositories

- https://github.com/AdExNetwork/adex-protocol/commit/4e5794bc837f69ee1741ff8c1ec5112edecf5197

Files

All solidity files included in the repository

Issues

AdEx Core

1. Rule that each state update has to be authored by `validator[0]` not enforced in contracts

In protocol specification (https://github.com/AdExNetwork/adex-protocol/tree/4e5794bc837f69ee1741ff8c1ec5112edecf5197#layer-2) it is declared that all states have to be authored by "lead validator" which is technically defined as address in the `validators` array of the `ChannelLibrary.Channel` atruct with index `0`. This requirement however isn't enforced in the state validation process implemented in the `AdExCore.channelWithdraw` function.

response

we acknowledge that this would be a clear improvement to safety, and we will implement it in a subsequent version: https://github.com/AdExNetwork/adex-protocol-eth/issues/68

2. Failure modes of the validator consensus protocol

Safety fault tolerance (consensus won't break if):

```
number of faulty validators <= 2 * ceil(2/3 * n) - n

or

number of faulty lead validators == 0
```

Liveness fault tolerance (consensus won't stall if):

```
number of faulty validators < n - ceil(2/3 * n)

and

number of faulty lead validators == 0
```

where `n` is the total number of validators

From the above, we can see that threshold for double spending is in the worst case 1/3 of validators and the lead validator. Since ownership of the funds can't be really conisdered final until they are withdrawn, we consider double spending to be a real risk. Especially in cases where number of validators > 2. This isn't fully appreciated in the documentation as ilustrated by statements such as:

Publishers have a constant guarantee that they can withdraw their latest earnings on-chain;

(https://github.com/AdExNetwork/adex-protocol/tree/4e5794bc837f69ee1741ff8c1ec5112edecf5197#flow)

This is not necessarily an issue to be fixed, rather a clarification of security assumptions.

response

Currently everything we do works with 2 validators and enforces 2 validators (except the on-chain contract); I'm pretty sure it's mentioned in the docs also, we're gonna amend the docs to include that a pBFT synchronization algorithm should be used if more than 2 validators are used.

3. Unnecessary restriction on who can call `channelWithdrawExpired` and `channelWithdraw` functions

since executing `channelWithdrawExpired` and `channelWithdraw` functions is always to the benefit of the recipient, the restriction that they can be only called by the recipient doesn't seem necessary. Making them callable by anyone on anybody else's behalf would also allow `Identity` contract to be simplified.

response

That's a very good point. When we were designing the Core, the Identity/relayer didn't exist, so it was between one additional input argument (the beneficiery) vs using `msg.sender`. We opted for `msg.sender`. Later on we came up with Identity/relayer which would've been helped by allowing anyone to call withdraw/`withdrawExpired`, but too many things already depend on the current fn signatures, so changing it is not worth it

AdEx Identity

4. Allowing relayer to open channels on user's behalf poses a significant risk

Ability to open and finance OUTPACE channels on behalf of the `Identity` owners allows the routine relayer to perform powerful DoS attacks and in collusion with at least one whitelisted validator even steal funds. This is a significant risk, especially if routine relayers are intended to be provided centrally.

response

Currently, all identities are deployed without setting a `registryAddr`, which means that a `channelOpen` routine op isn't possible; furthermore, the docs will be updated to reflect this added risk and we will consider removing it altogether if it isn't used.

5. Same validator can be added by the relayer multiple times, reducing validator/relayer collusion treshold

At first look, routine relayer needs to collude with two whitelisted validators to be able to steal tokens from the `Identity` contract, the fact he can add the same validator multiple times however lowers this threshold to 1.

response

See response to #4

6. Minor griefing opportunities in IdentityFactory

Anybody can frontrun owner generated call of `deployAndFund` by either calling `deploy` or `deployAndExecute` making the owner's transaction fail. This forces the factory owner to fund the Identity contract in a different way, forcing additional transaction costs.

response

Acknowledged; we will consider ways of fixing this, tracked in: https://github.com/AdExNetwork/adex-protocol-eth/issues/70

7. `WitdrawTo` authorisation should probably rank below `Transactions` authorisation

Since anybody with the `Transaction` authorisation has a complete control over the `Identity` contract, including setting authorisation levels, `WithdrawTo` is a practically redundant authorisation level (besides serving as a whitelisting device for relayer initiated withdraws). Putting it under `Transactions` authorisation would make the authorisation system more expressive.

response

The right thing to do is probably to re-work the system using bitmaps, to allow for fine-grained control of privileges. The reason [we have put `WithdrawTo` privilege level above `Transactions`] is, we don't want `Transactions` to also imply `WithdrawTo`, but your argument that `WithdrawTo` should not imply Transactions is really solid. In general, having each privilege level imply all the lower ones too is suboptimal.

Conclusion

We haven't discovered any critical issues during the audit. Solutions implemented and proposed by the developers address our findings sufficiently.