Smart Contract Audit

Arbitrable Directory

By,

Shebin John

15th July 2020

Index

Note: Some threat levels or headings might be empty if there is no vulnerability/updates/suggestions found.

Introduction

The contract audited here is/are:
https://github.com/unknownunknown1/org.id-directories/blob/a78c009/contracts/ArbitrableDirectory.sol
Related Contracts:
https://github.com/windingtree/lif-token/blob/00b808d/contracts/LifToken.sol
https://github.com/windingtree/org.id/blob/5850d02/contracts/OrgIdInterface.sol
https://github.com/windingtree/org.id/blob/5850d02/contracts/OrgId.sol
and was shared by Kleros for auditing purposes while working with them.

Based on the audit, we were able to find 3 High threats, no Medium threat, and 1 Low threat with some other changes in the optimization, readability, typos, and comments section.

High Threats

1. The token contract used in this is not decentralized as the owner has the right to burn tokens from any address they wish. He won't be able to transfer the token to his address, but deflation can increase the price. Only if the minting is finished, the owner loses this right.
   For this particular instance of token contract, the minting is finished: https://etherscan.io/token/0xeb9951021698b42e4399f9cbb6267aa35f82d59d#readContract

2. Due to the upgradeable contract nature for both OrgId and ArbitrableDirectory, if the owners (in this case, I believe it is a multisig, so owners) of the contract go rogue by adding malicious contract, can result in any vulnerability based on that smart contract.

   Even if they are doing the upgrade process by DAO, if by any chance, they find a vulnerability, a public DAO vote for patching it up can lead the hackers to see the vulnerability in the proposed patch.

3. According to getOrganization() the fetching in requestToAdd() is wrong. The same bug is in Line 519, 546 and 650.

Medium Threats

1. None

Low Threats

1. This vulnerability only applies if we allow a previously removed entry to give another try. Considering the requester made a request to add and was challenged by an attacker. The requested understood the mistake and waited for the responseTimeout to end so that the attacker (thought as a genuine user by the requester) can win. But after the responseTimeout, if the attacker refuses the token transfer here (he might have done the challenge using a smart contract and now have destroyed it using self destruct), then it basically makes the requester to go to high troubles to create a new organization once again, rather than applying it with the same organization id.

Optimization & Readability

1. I believe _getOrganizations() is over-engineered. It is an internal function used to get either all in/active organizations present or get all in/active units of an organization. Two separate functions (getOrganizations() and getUnits()) are written which calls this function with parameters for each case. If we write the logic separately to get the data in those two functions itself, we could avoid a lot of checks done in a for loop. In a view, it might seem unnecessary, but if this is used in other functions which is not a view, then the gas cost can increase a lot, based on the number of organizations added. For each organization, there are 6 checks inside the for loop.

Typo's & Comments

1. * @param solt Unique hash required for **organization** identifier creation
2. * @param solt Unique hash required for **organization unit** identifier creation
3. // Party **by** default when there is no challenger or requester. Also used for unconclusive ruling.
4. @dev Get all organizations' ORGiD hashes (as it can also contain inactive ones if includeInactive is set to true). Similar case in Line 435 as well.
5. In line 92 it might be informational if we mention that the deposit will be done in ETH.

Suggestions

1. None