Q ID Internal Audit

July 2023.

Audited by: Arvolear.

Disclaimer

A smart contract audit does not guarantee absence of bugs, vulnerabilities, and exploits. Subsequent audits and bug bounties are highly recommended.

About Q ID

Q ID is a solution that provides identity management to the Q blockchain. Q ID is based predominantly on IDEN3 protocol and has the following componets: solidity smart contract, circom circuits, and go backend.

Issuers are able to issue **claims** that state certain properties about the account they are bound to. Claims are further aggregated into **identities** that can be verified on-chain through smart contracts. The claim issuance itself results in the **state transition** that occurs on-chain via zero-knowledge proofs.

Anyone can issue claims, however Q ID trusts only the whitelisted issuers.

Audit Summary

We identify the protocol as **not safe** as there is one **critical** issue that was found during the audit.

Severity classification

- Optimization - The issue suggests how to optimize a smart contract.
- Note - The issue requires attention, yet does not pose any risks.
- Low - The issue has minimal impact on the contract's ability to operate.
- Medium - The issue affects the contract's operability and does not lead to a loss of funds.
- High - The issue leads to a loss or incorrect allocation of funds.
- Critical - The issue leads to a straightforward exploit.

Analysis Summary

Review commit hash - `7845954fe274be0adacf22dde800b83bdc1a1a0f`

Scope

Smart contracts in scope of the audit:

- contracts/validators/QueryMTPValidatorV2.sol

- contracts/validators/QuerySigValidatorV2.sol
- contracts/validators/QueryValidatorV2.sol
- contracts/core/StateV2.sol
- contracts/core/ZKPQueriesStorage.sol
- contracts/core/QDAOMembershipManager.sol
- contracts/core/KYCHashStorage.sol

Number of issues found, categorized by their severity:

- Critical: 1 issue
- High: 0 issues
- Medium: 0 issues
- Low: 1 issue
- Note: 3 issues
- Optimization: 1 issue

Findings

QDAOMembershipManager [C-01] Critical - no ownership initialization

Description

The contract is upgradeable and there is no `__Ownable_init()` function call in the initializer. This does not lead to the loss of funds or vital information, however this issue make the system completely unusable after deploy.

Recommendation

Initialize the owner properly.

StateV2 [L-01] Low - `transitStateWithCommit()` may lead to wrong calculation of claims

Description

The `for` loop in the `transitStateWithCommit()` may be stopped via `break` if wrong `claimType` is passed first. This might lead to `_schemaInfo.claimsCount` miscalculation.

```
for (uint256 i = 0; i < MAX_CLAIM_COMMIT_COUNT; i++) {
    ...

    if (currentClaimType_ == ClaimTypes.ADDED_CLAIM) {
        ...

        _schemaInfo.lastSerialNumber = currentSerialNumber_;
        _schemaInfo.claimsCount += 1;
    } else if (currentClaimType_ == ClaimTypes.REVOKED_CLAIM) {
        ...

        _schemaInfo.revokedClaims.add(currentSerialNumber_);
```

```
        _schemaInfo.claimsCount -= 1;
    } else {
        break;
    }
}
```

Recommendation

Use `continue` instead of `break`.

StateV2 [O-01] Optimization - `claimsCount` may be removed

Description

In order to calculate claims, 3 entities are used:

- lastSerialNumber
- claimsCount
- revokedClaims

Instead of explicitly calculating `claimsCount`, you can assume it to be equal to `lastSerialNumber - revokedClaims.length`.

Recommendation

Remove the claimsCount variable and calculate it implicitly.

QDAOMembershipManager [N-01] Note - typo

Description

Variable `storedCicuitQuery_` is misspelled

Recommendation

Rename to `storedCircuitQuery_`

IKYCHashStorage [N-02] Note - event parameter can be indexed

Description

The `KYCHashUpdated()` event has to indexed parameters, however it might be convenient to index the `kycHash` parameter so 3'rd party services can track user updates quickly.

Recommendation

Index `kycHash` parameter.

StateV2 [N-03] Note - is Ownable2StepUpgradeable really needed?

Description

Other contracts descend from regular `OwnableUpgradeable` contract, however StateV2 inherits `Ownable2StepUpgradeable`. Is this desired?

Recommendation

Use regular `OwnableUpgradeable` contract to be consistent.