CD SECURITY

AUDIT REPORT

DORA - Solidity

July 2024

Introduction

A time-boxed security review of the **DORA 2.0** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **DORA 2.0**

DORA means Distributed Oracle Agreement. The protocol aims to bridge the gap between the blockchain and the off-chain by providing reliable data to the blockchain. An important note is that the off-chain functionality of DORA was NOT in the scope of this audit. Here is how the workflow of the protocol:

The audit conducted this time focuses on an updated version of DORA.

Documentation from our last report can be found here.

Full documentation here.

Severity classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - bbef0af8b7dad07fdb16c05daa0a1e050900f1f7**

***fixes review commit hash* - 1d34b67defeb33402303c3583aa39401a6be3be5**

Scope

The following smart contracts were in scope of the audit:

- `solidity/svalue/with_verification/decentralized/src/*`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issue
- Medium: 0 issues
- Low: 2 issues
- Informational: 3 issues

Findings Summary

ID	Title	Severity
[L-01]	Native tokens can't be sent with transactions	Low
[L-02]	Disable initializers in upgradable contracts	Low
[I-01]	Unstable pragma statement	Informational
[I-02]	Use of named mapping parameters	Informational
[I-03]	Unused events and typo	Informational

Detailed Findings

[L-01] Native tokens can't be sent with transactions

In `MultiSignatureWallet::submitTransaction`, there is a `_value` input parameter which is supposed to be the amount of ether to be sent with the transaction. The transaction is then pushed into an array with the according value and needs to be confirmed by the other owners.

```
    function submitTransaction(
         address _to,
         uint256 _value, <@audit
         uint64 _timeoutDuration,
         bytes memory _data
     ) external {
         onlyOwner(msg.sender);
         uint256 txIndex = transactions.length;

         transactions.push(
             Transaction({
                 ....
                 value: _value,
                 data: _data
             })
         );
```

The problem is that the function is not payable and ether can't be sent with the transactions. This means that any value can be passed as an input parameters and the transaction will be submitted without providing any native tokens.

Recommendations

Make the `submitTransaction` method `payable` and use `msg.value` instead of `value`.

Client

Fixed

[L-02] Disable initializers in upgradable contracts

Avoid leaving a contract uninitialized.

An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the `_disableInitializers` function in the constructor to automatically lock it when it is deployed.

Client

Fixed

[I-01] Unstable pragma statement

The `MultiSignatureWallet` contract is currently using a floatable `^0.8.24` pragma statement. It is recommended as a best practice to use a stable pragma statement to ensure that the contract is compiled with a specific, tested version of the Solidity compiler.

Client

Fixed

[I-02] Use of named mapping parameters

Since version `0.8.18`, parameters inside mappings can be named. Example:

```
mapping(string object => uint value) public objects;
```

Consider using named mappings for better readability.

Client

Fixed

[I-03] Unused events and typo

There are some unused errors tha can be removed:

```
    error PairIdIsAbsentForHCC();
    error PairIdsAreAbsentForHCC();
    error DerivedPairNotAvailable(uint256 pairId);
    error ArrayLengthMismatched();
    error InvalidDerivedPair();
```

There is typo in the name of the following function - `updateNumConfimations`.

Client

Fixed