HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: CLABS
Date: July 29th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for CLABS
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	ERC20; Attestation; EsccountsPerIssuerrow
Platform	EVM
Network	Ethereum,
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	-
Timeline	13.06.2022 - 29.07.2022
Changelog	12.07.2022 - Initial Review 29.07.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by CLABS (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/celo-org/celo-monorepo/tree/ASv2/packages/protocol
Commit:
d3322f46d877db11ee705f22e7fb103085d10301
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://docs.celo.org/celo-codebase/protocol/transactions/escrow
https://clabsco.notion.site/Federated-Attestation-Protocol-ASv2-05dc4843139842768ad9fe192cb46c00

Type: Technical description
https://github.com/celo-org/celo-monorepo/pull/9560
https://github.com/celo-org/celo-monorepo/pull/9631
https://github.com/celo-org/celo-monorepo/pull/9636

Type: Functional requirements
https://docs.celo.org/celo-codebase/protocol/transactions/escrow

Integration and Unit Tests: Yes
Deployed Contracts Addresses: -
Contracts:
File: ./packages/protocol/contracts/common/Initializable.sol
SHA3: ffac4ba806b1e411ad99b6757489672f8a1baa6c9c5fb84f40a310a5be5cacc3

File: ./packages/protocol/contracts/common/libraries/ReentrancyGuard.sol
SHA3: 4c4d2516e557392225dc4ecfab1e4fa1c41e39bd06d27bbdf7f777c4b04ac1ab

File: ./packages/protocol/contracts/common/Signatures.sol
SHA3: 1863f547bb7939b191f3078ea901ecb73531259fd98a51e93f71c3f2a4026a29

File: ./packages/protocol/contracts/common/UsingPrecompiles.sol
SHA3: 485abfc8a6e98e50198a805d875775fb018b49b76d23e30eaa94b97503061c6d

File: ./packages/protocol/contracts/common/UsingRegistryV2.sol
SHA3: a87d1b18dc4c975e18bb5c9e40491b16e67898444b34ce785df34bed28a029b8

File:
./packages/protocol/contracts/common/UsingRegistryV2BackwardsCompatible.sol
SHA3: 89749df81c2e34a606b6ba87e1ac48f415137517d73a7c9da7b6a8dcbd9381a0

File: ./packages/protocol/contracts/identity/Escrow.sol
SHA3: e970ed6cf2d00b4bbdeb954d281e508b52fee8dc1c99d2978787ab2904d89575

File: ./packages/protocol/contracts/identity/FederatedAttestations.sol
SHA3: 048773e19c15e288d6bd454f91126135dec48cde49b771a4dc5653d08f2218aa

Second review scope

Repository:
https://github.com/celo-org/celo-monorepo/tree/ASv2/packages/protocol
Commit:
c1bd057484c4601436eb35c563d431f80d5f94b6
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://docs.celo.org/celo-codebase/protocol/transactions/escrow
https://clabsco.notion.site/Federated-Attestation-Protocol-ASv2-05dc4843139842768ad9fe192cb46c00

Type: Technical description
https://github.com/celo-org/celo-monorepo/pull/9560
https://github.com/celo-org/celo-monorepo/pull/9631
https://github.com/celo-org/celo-monorepo/pull/9636

Type: Functional requirements
https://docs.celo.org/celo-codebase/protocol/transactions/escrow

Integration and Unit Tests: Yes
Deployed Contracts Addresses: -
Contracts:
File: ./packages/protocol/contracts/common/Initializable.sol
SHA3: ffac4ba806b1e411ad99b6757489672f8a1baa6c9c5fb84f40a310a5be5cacc3

File: ./packages/protocol/contracts/common/libraries/ReentrancyGuard.sol
SHA3: 4c4d2516e557392225dc4ecfab1e4fa1c41e39bd06d27bbdf7f777c4b04ac1ab

File: ./packages/protocol/contracts/common/Signatures.sol
SHA3: 1863f547bb7939b191f3078ea901ecb73531259fd98a51e93f71c3f2a4026a29

File: ./packages/protocol/contracts/common/UsingPrecompiles.sol
SHA3: 485abfc8a6e98e50198a805d875775fb018b49b76d23e30eaa94b97503061c6d

File: ./packages/protocol/contracts/common/UsingRegistryV2.sol
SHA3: 89749df81c2e34a606b6ba87e1ac48f415137517d73a7c9da7b6a8dcbd9381a0

File:
./packages/protocol/contracts/common/UsingRegistryV2BackwardsCompatible.sol
SHA3: 89749df81c2e34a606b6ba87e1ac48f415137517d73a7c9da7b6a8dcbd9381a0

File: ./packages/protocol/contracts/identity/Escrow.sol
SHA3: 8884142aba73416fa44e8b53b3de56abfc6212b89c6d6633b25981f3e6e6cf3e

File: ./packages/protocol/contracts/identity/FederatedAttestations.sol
SHA3: 7241eca1d0de17a2a512c051c80fec5bb2322d4a65eb2bd609a343b09ca1e40c

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. Technical and functional documents are provided.

Code quality

The total CodeQuality score is **8** out of **10**. The code mostly follows official guidelines but uses an outdated version. The tests were provided, and they are executing without a problem.

Architecture quality

The architecture quality score is **7** out of **10**. The project has clear, clean architecture. The development environment is not well configured.

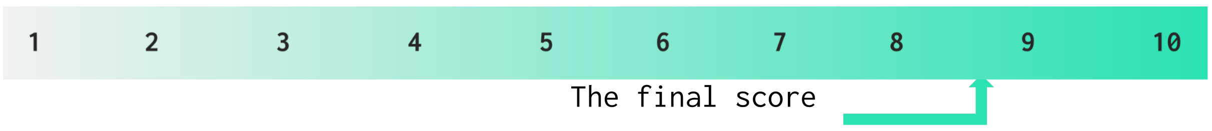
Security score

As a result of the audit, the code contains **1** medium and **4** low severity issues. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.8**.



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Failed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Failed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Failed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Not Relevant
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Failed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Not Relevant
Authorization	SWC-115	tx.origin should not be used for	Not Relevant

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of	Failed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

A federated attestation protocol(v2 in scope) leverages a federated set of independent issuers to perform phone number attestations. It has an escrow functionality that allows users to send payments to other users.

- *FederatedAttestations* – its main purpose is to map attestations to accounts. It allows users to register or revoke attestations.
- *Escrow* – allows users to send payments to other users. These payments can be withdrawn by the receiver, or they will expire.
- *Initializable* – a contract that implements the check if a contract is already deployed.
- *Signatures* – a library that handles signature operations.
- *UsingPrecompiles* – a helper contract that mainly handles block and header-related data.
- *ReentrancyGuard* – a library that helps contracts guard against reentrancy attacks.
- UsinRegistryV2 – registry contract.
- UsinRegistryV2BackwardsCompatible - contains a different implementation of the getVersion contract.

Privileged roles

- Owner: The owner of the Escrow contract:
  - can add a new default trustedIssuers
  - can remove an existing default trustedIssuers

Risks

- ERC20 tokens being used in the competitions should adopt standard ERC20 implementations. Any implementation with a fallback logic can lead to unexpected behavior during token transfers, such as reverts. This can lead to potential Denial-of-Service vulnerabilities.

- The repository contains contracts that are out of the audit scope. The secureness of such contracts could not be guaranteed.

Findings

■■■■ Critical

No critical issues were found.

■■■ High

1. Users can withdraw payments after expiry

The system allows payments to be withdrawn after their expiry duration has passed.

This will allow users to withdraw expired payment records.

File: ./contracts/identity/Escrow.sol

Contract: Escrow

Function: withdraw

Recommendation: This feature is not documented correctly. The documentation should be updated.

Status: Fixed.Link

■■ Medium

1. Out-of-Gas possibility

Iterating over large structures (user-supplied or not) and performing external calls in loops may lead to out-of-Gas exceptions. FederatedAttestations contract iterates over user-supplied arrays.

This can lead to out-of-Gas exceptions.

File: ./contracts/identity/FederatedAttestaions.sol

Contract:FederatedAttestations

Functions: batchRevokeAttestations

Recommendation: Implement array size limitations.

Status: Reported. (Related Customer Answer)

2. Unfinished code

The provided code should be implemented in the full logic of the project. Since any missing parts, TODOs, or drafts can change in time, the robustness of the audit cannot be guaranteed.

File: ./contracts/identity/FederatedAttestaions.sol

Contract: FederatedAttestations

Functions: registerAttestationAsIssuer, registerAttestation

Recommendation: Complete the code to meet all the requirements and delete the TODO comments.

Status: Fixed (c1bd057484c4601436eb35c563d431f80d5f94b6)

3. **Required SafeERC20 implementation**

   Tokens in the ERC20 standard, return a boolean due to the transfer method, but it is important to use the SafeERC20 library to support tokens that do not conform to the ERC20 standard and do not return booleans.

   This issue makes it impossible to support tokens that do not conform with the ERC20 standard.

   File: ./contracts/identity/Escrow.sol

   Contract: Escrow

   Function: transfer

   Recommendation: Implement SafeERC20 library.

   Status: Fixed (c1bd057484c4601436eb35c563d431f80d5f94b6)

■ Low

1. **Outdated Solidity version**

   Using an outdated compiler version can be problematic, especially if publicly disclosed bugs and issues affect the current compiler version. The project uses compiler version 0.5.13.

   Files: ./contracts/common/UsingRegistryV2.sol

   ./contracts/common/Signatures.sol

   ./contracts/common/ReentrancyGuard.sol

   ./contracts/common/Initialize.sol

   ./contracts/common/UsingPrecompiles.sol

   ./contracts/common/UsingRegistryV2BackwardsCompatible.sol

   ./contracts/identity/FederatedAttestaions.sol

   ./contracts/identity/Escrow.sol

   Contracts: UsingRegistryV2, Signatures, Initialize, UsingPrecompiles, UsingRegistryV2BackwardsCompatible, Federated Attestations, Escrow

   Function: -

   Recommendation: Use a contemporary compiler version.

   Status: Reported. The development environment can be configured to work with multiple compiler versions. (Related Customer Answer)

2. **Floating pragma**

Unlocked pragmas may cause the contract to be deployed with a different Solidity version from the tested. The project uses floating pragmas ^0.5.13.

This can lead to encountering undiscovered bugs.

Files: ./contracts/common/UsingRegistryV2.sol

./contracts/common/Signatures.sol

./contracts/common/ReentrancyGuard.sol

./contracts/common/Initialize.sol

./contracts/common/UsingPrecompiles.sol

./contracts/common/UsingRegistryV2BackwardsCompatible.sol

./contracts/identity/FederatedAttestaions.sol

./contracts/identity/Escrow.sol

Contracts: UsingRegistryV2, Signatures, ReentrancyGuard, Initialize, UsingPrecompiles, UsingRegistryV2BackwardsCompatible, FederatedAttestaions, Escrow

Function: -

Recommendation: Lock pragma to a specific compiler version.

Status: Reported. (Related Customer Answer)

3. **Usage of state variables default visibility**

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Files: ./contracts/common/UsingPrecompiles.sol

Paths: *TRANSFER, FRACTION_MUL, PROOF_OF_POSSESSION, GET_VALIDATOR, NUMBER_VALIDATORS, EPOCH_SIZE, BLOCK_NUMBER_FROM_HEADER, HASH_HEADER, GET_PARENT_SEAL_BITMAP* and *GET_VERIFIED_SEAL_BITMAP*

Contracts: UsingPrecompiles

Recommendation: Explicitly define visibility for all state variables.

Status: Reported. (Related Customer Answer)

4. **Unused variables**

There are variables in UsingPrecompiles and UsingRegistryV2 contracts that are defined but never used.

Files: ./contracts/common/UsingRegistryV2.sol

./contracts/common/UsingPrecompiles.sol

Contracts: UsingRegistryV2, UsingPrecompiles

Paths: *TRANSFER, DOWNTIME_SLASHER_REGISTRY_ID, DOUBLE_SIGNING_SLASHER_REGISTRY_ID, GOVERNANCE_SLASHER_REGISTRY_ID*

Recommendation: Remove unused variables.

Status: Mitigated. UsingRegistryV2 is useful for other contracts that want to interact with contracts in the registry, and it would be helpful to access those constants to find the proper addresses for DowntimeSlasher. (Related Customer Answer)

5. **Functions that could be declared external**

minQuorumSizeInCurrentSet, minQuorumSize, getVerifiedSealBitmapFromHeader, getParentSealBitmap, hashHeader, getBlockNumberFromHeader, checkProofOfPossession, numberValidatorsInCurrentSet, validatorSignerAddressFromSet, validatorSignerAddressFromCurrentSet, getEpochNumber and *fractionMulExp* functions of UsingPrecompiles and *getSignerOfAddress* and *getSignerOfTypedDataHash* functions of signatures can be declared external.

Files: ./contracts/common/UsingPrecompiles.sol

./contracts/common/Signatures.sol

Contracts: UsingPrecompiles, Signatures

Functions: minQuorumSizeInCurrentSet, minQuorumSize, getVerifiedSealBitmapFromHeader, getParentSealBitmap, hashHeader, getBlockNumberFromHeader, checkProofOfPossession, numberValidatorsInCurrentSet, validatorSignerAddressFromSet, validatorSignerAddressFromCurrentSet, getEpochNumber, fractionMulExp, getSignerOfAddress, getSignerOfTypedDataHash

Recommendation: Convert to external. Making functions external instead of public reduces Gas costs during execution.

Status: Reported. (Related Customer Answer)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.