SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Tokeny, OnchainId
Date: March 21, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Tokeny
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	Personal Id, Access Control
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.onchainid.com/
Changelog	19.01.2023 - Initial Review 16.02.2023 - Second Review 21.03.2023 - Third Review

Table of contents

Introduction **5**
Scope **5**
Severity Definitions **8**
Executive Summary **9**
Checked Items **10**
System Overview **13**
Findings **15**
Critical 15
C01. Data Consistency 15
High 16
H01. Denial of Service 16
H02. Requirements Violation 17
H03. Requirements Violation 17
H04. Requirements Violation 17
Medium 18
M01. Inconsistent Data - Unused Return Value 18
M02. Contradiction - Missing Validation 18
M03. Best Practice Violation - Checks-Effects-Interactions Pattern 18
M04. Unscalable Functionality - Code Duplication 19
M05. Inefficient Gas Model - Storage Abuse 19
M06. Inefficient Gas Model - Storage Abuse 20
M07. Inefficient Gas Model - Storage Abuse 20
M08. Inefficient Gas Model - Storage Abuse 20
M09. Best Practice Violation - Missing Validation 21
M10. Best Practice Violation - Missing Validation 21
M11. Unscalable Functionality 21
M12. Contradiction - NatSpec Comment Contradiction 21
M13. Best Practice Violation - Lock of Native Tokens 22
M14. Inconsistent Data - Incorrect Event Emitting Order 22
M15. Inefficient Gas Model - Storage Abuse 23
Low 23
L01. Unused Import 23
L02. Missing Zero Address Validation 23
L03. Redundant Check 24
L04. Unspecific Storage 24
L05. Unclear Error Message 24
L06. Gas Optimization in Comparison 25
L07. Style Guide Violation 25
L08. Naming Consistency 25
L09. Functions that Can Be Declared External 26
L10. Style Guide - Redundant Code 26
L11. Best Practice Violation - Shadowing State Variable 26
L12. Unfinished NatSpec 26
L13. State Variables that Could Be Declared as Immutable 27
L14. Redundant Variable 27
L15. Redundant Require Statement 27

Disclaimers 28

Introduction

Hacken OÜ (Consultant) was contracted by Tokeny (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/onchain-id/solidity/tree/hacken-audit
Commit	31209d8db055025c329890da8034d7285ea923c3
Whitepaper	Not Provided
Functional Requirements	https://docs.tokeny.com/docs/general-concept
Technical Requirements	https://docs.tokeny.com/docs/general-concept
Contracts	File: ./contracts/ClaimIssuer.sol SHA3:211e7295edfaa9c8dede830a419774b1fc2f3949566c12bca2d2229636aa71b7 File: ./contracts/factory/IdFactory.sol SHA3:30dc5d149cba5b0d1a054c1537d4098e6ed473e90d39590042dd0ea48d34070f File: ./contracts/factory/IIdFactory.sol SHA3:855a27c9850e172a156dc76700073929b7268e403396bf5af95f0465ea5fb50b File: ./contracts/Identity.sol SHA3:5ba66dc8fd94c60e85f97b58ecbe0fc88b7f18fa9e521da63bdbf52684ba0fbb File: ./contracts/interface/IClaimIssuer.sol SHA3:5d67a4f2619908240ff8acb14ba7100fd895e39173856c84056ac3ff8f0aee68 File: ./contracts/interface/IERC734.sol SHA3:3b9b805431f35e435f61c9880371c11991cff830a1b201231e337788a7982942 File: ./contracts/interface/IERC735.sol SHA3:8e16b1642f649e0007fe5b781505655bdc26d62d7533105dcba559c089ac040b File: ./contracts/interface/IIdentity.sol SHA3:b53aa1d640be24bb79e7b9d71858792fbcb4f8efa96610bd19ea22edc350b933 File: ./contracts/interface/IImplementationAuthority.sol SHA3:288f385822ca50b6a84ecdb4d9ad1562690a43c227c828db1b7f9fb1ef66699d File: ./contracts/proxy/IdentityProxy.sol SHA3:f7e369ccb23626cd11501522bfe6f792ce7ce7405683465d7cd93021e62e8221 File: ./contracts/proxy/ImplementationAuthority.sol SHA3:91870097a5c6d0e9697ed2a7c06b3cb5c5ea5ec710c7d66c3a6f37a5f8c64edb File: ./contracts/storage/Storage.sol SHA3:58944a4bc2ebf864b55b6e607eaaf46a5069c0a960ddaa99ae17a5266f156fc8 File: ./contracts/storage/Structs.sol

	SHA3:1c6dd62e547527e8045d721d96e9400ef5ff4e22fbd4b5187134c6c85c714d84 File: ./contracts/version/Version.sol SHA3:f7e62de2220c57dc5349848dcf36da48d4d48e027c77053fbf51edbf7558bf45

Second review scope

Repository	https://github.com/onchain-id/solidity/tree/5c40bc7c7b462e4a4bff7925925b0759f7720442
Commit	5c40bc7
Contracts	File: ./contracts/ClaimIssuer.sol SHA3:25a54aab0f388d3a978a22f0fc2aa36ad5e5adee7f39e5381215d6f9c8cb9999 File: ./contracts/factory/IdFactory.sol SHA3:1504ffdf7299d49f689f1f40ce039e0f0cebd937b724ec5636f9d71ab5d71c8f File: ./contracts/factory/IIdFactory.sol SHA3:2d8713abb0679395282204b5205eae249b5b2004588ac6ab78dda513bd207a5e File: ./contracts/Identity.sol SHA3:b72fc074c6c9e64c435edcb1bbb67a111f117468c94e3a1dd3e5b576e639dd62 File: ./contracts/interface/IClaimIssuer.sol SHA3:5d67a4f2619908240ff8acb14ba7100fd895e39173856c84056ac3ff8f0aee68 File: ./contracts/interface/IERC734.sol SHA3:1de188f676b87e1d3fdfe406117bc8490c5a4c5529a1dc7a2f92b540eff44281 File: ./contracts/interface/IERC735.sol SHA3:bed91d3ab3dad913c3665b13a5f897af0fc942e8c6b45008357aeb403c273e84 File: ./contracts/interface/IIdentity.sol SHA3:b53aa1d640be24bb79e7b9d71858792fbcb4f8efa96610bd19ea22edc350b933 File: ./contracts/interface/IImplementationAuthority.sol SHA3:288f385822ca50b6a84ecdb4d9ad1562690a43c227c828db1b7f9fb1ef66699d File: ./contracts/proxy/IdentityProxy.sol SHA3:9069478e71e058eb1bf30d34310db1d739b12adc6ddd74a8df78d033411dc255 File: ./contracts/proxy/ImplementationAuthority.sol SHA3:b675a13d47da177bf49dbf6d6c25fca508eaab64c3cefb4aa6a0aff92f9e5e2b File: ./contracts/storage/Storage.sol SHA3:d5f16b2c7c6666a7eea0349d6525c995ff1399860028254a387e8d5aea297cc5 File: ./contracts/storage/Structs.sol SHA3:1c6dd62e547527e8045d721d96e9400ef5ff4e22fbd4b5187134c6c85c714d84 File: ./contracts/version/Version.sol SHA3:ad505d3aca038c468a75d7edd05af9e992d685f8e18cb3c5c37538c79346905c

Third review scope

Repository	https://github.com/onchain-id/solidity/tree/c20c1480b5e2ad10d3ffd23d9d760f69f1645825
Commit	c20c148

Contracts	File: ./contracts/ClaimIssuer.sol SHA3:10c80034acc89133aefb577bf798b8206a239927acfff9036241f57f6d9ee747 File: ./contracts/factory/IdFactory.sol SHA3:602c92087cd3591d9e7be542f13d6627b45ca415007a3094b84460ae80b73e6c File: ./contracts/factory/IIdFactory.sol SHA3:2d8713abb0679395282204b5205eae249b5b2004588ac6ab78dda513bd207a5e File: ./contracts/Identity.sol SHA3:74ea9ade0801e1da9781872a2cbcf76b527591427f0a82d6a9bf600aed5e7b0c File: ./contracts/interface/IClaimIssuer.sol SHA3:57d28cba48929f1495dcfef2c52da870f0868c1d38136951ae16839b05d8486c File: ./contracts/interface/IERC734.sol SHA3:57cafb3f58155e7fd74d0fe8ab38bf073dd7ba71430b5b214eb9db4c3e3ef771 File: ./contracts/interface/IERC735.sol SHA3:774d17fb40f7088c6acd95aa7f43c0186c4f485794922f666e5926fba17f11c5 File: ./contracts/interface/IIdentity.sol SHA3:b53aa1d640be24bb79e7b9d71858792fbcb4f8efa96610bd19ea22edc350b933 File: ./contracts/interface/IImplementationAuthority.sol SHA3:288f385822ca50b6a84ecdb4d9ad1562690a43c227c828db1b7f9fb1ef66699d File: ./contracts/proxy/IdentityProxy.sol SHA3:e58c53e570e2eb49c9095c595484038a088b96b920731809a6afd4498b679953 File: ./contracts/proxy/ImplementationAuthority.sol SHA3:5e1181d9fd9f776caca89a5626af592ed8cb75d0217846246262e50e941cceb5 File: ./contracts/storage/Storage.sol SHA3:1cfa2668656be14d27347c2e68ebb195176ffec9517a9876898eea9962b8540f File: ./contracts/storage/Structs.sol SHA3:1c6dd62e547527e8045d721d96e9400ef5ff4e22fbd4b5187134c6c85c714d84 File: ./contracts/Test.sol SHA3:885ad5cb350ce74476f2a4c5716494fbb8bf9df167531d9a710843b78a9fc6ff File: ./contracts/version/Version.sol SHA3:ad505d3aca038c468a75d7edd05af9e992d685f8e18cb3c5c37538c79346905c

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are partially missed.
- NatSpec is consistent.
- There is no apparent flow of the contracts.
- Run instructions are provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Best practices are followed.

Test coverage

Test coverage of the project is **89.77%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** 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: **9.4**.

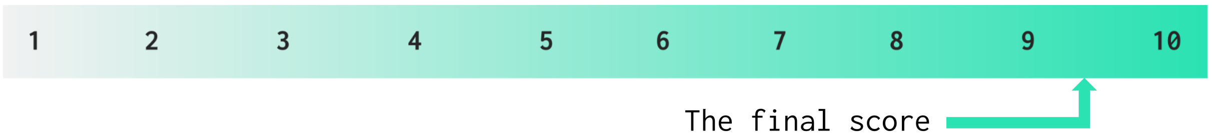

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
19 January 2023	13	13	4	1
16 February 2023	3	4	2	1
21 March 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items 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.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
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.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
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 identifiers 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.	Passed
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.	Not Relevant
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.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

ONCHAINID is a blockchain-based identity system that identifies individuals and organizations, allowing them to enforce compliance and access digital assets.

ONCHAINIDs are self-sovereign identities. This means that each holder of an identity is in control of their information and who has access to it.

Their address is a unique identifier that can safely be used by a service provider to identify their owner, and even to sign requests such as direct authentication on a website.

The project aims to achieve a means of decentralized authentication that is in control of the identity holders themselves.

The project consists of standards for claim holders and key holders, which are then combined to create identities.

Information about the contracts in the scope;

- **Identity.sol:** The contract that implements the IERC734 and IERC735 standards and combines them to create an identity which can add/remove keys and claims.
- **IdFactory.sol:** The contract that handles the deployment of identity proxies.
- **ClaimIssuer.sol:** The contract that represents the claim issuers by extending Identity.sol.
- **IdentityProxy.sol:** Proxy contract for upgradeability in Identity.
- **Structs.sol:** The contract that holds struct variables for upgradeability.
- **ImplementationAuthority.sol:** The contract responsible for storing and updating the implementation logic of upgradeable contracts.
- **Storage.sol:** The contract that holds variables for upgradeability.
- **IIdFactory.sol:** Interface for IdFactory.
- **Version.sol:** The contract to hold the version of the Identity contract.
- **IERC734.sol:** Key holder standard interface.
- **IERC735.sol:** Claim holder standard interface.
- **IClaimIssuer.sol:** Interface for ClaimIssuer.
- **IIdentity.sol:** Interface for Identity.
- **IImplementationAuthority.sol:** Interface for ImplementationAuthority.

Privileged roles

- Owner: The owner of the system. Responsible for the upgrading of the contracts.
- Claim Issuer: Responsible for issuing identity holders’ claims on their identity.

- Identity Owner: The OnchainId identities user.

Risks

- The upgradeable nature of the contracts puts the implementation at risk in case of logic upgrade.
- The EIP-734 and EIP-735 were closed and not integrated into the Ethereum standards. The project is building upon those standards.
- The Identity.sol contract tries to implement the old EIP-725 standard, which was updated to the 725v2.

Findings

■■■■ Critical

C01. Data Consistency

In Identity.execute(), the input _value is not checked so that it is equal to the msg.value. The documentation does not specify this, but the function behavior suggests this.

This may lead to unwanted behavior and leave excess native tokens stuck in the Identity contract if a users parameter _value is less than their msg.value. Additionally, a user can use an input _value higher than their msg.value, resulting in a loss of funds of the Identity contract when the transaction will be approved and executed.

Path:
./contracts/Identity.sol: execute(), approve()

Recommendation: Either check the parameter _value so that it is equal to msg.value or pass msg.value directly into _executions[_executionNonce].value.

Status: Mitigated (

commit *5c40bc7*, Customer notice:

“The execute and approve method should be able to utilize any native token stored at the identity contract address. The ability to call execute with a value allows to retrieve this value from the contract, or to use it when calling another contract without using the funds of the sender of the transaction (this is by extension a way to retrieve any native token “stuck” in the identity contract). This is expected behavior.”

However, this may still cause a vulnerability, as stated in the issue, which leads to the loss of funds for the Identity contract, as there is no restriction on the execute() function caller.

An attacker can create an unlimited number of execute() transactions with malicious data, for example, to transfer stored funds to the attacker's address.

All those transactions will be shown in the frontend application, "Dapp/Mobile app", and the identity owner or key signer can be tricked into approving such transactions. In short, the end user can be spammed with requests.

commit *c20c148*, Customer notice:

“The ability for all to call the `execute()` method is a core functionality, an attacker could indeed send a lot of `execute` request to the Identity contract (paying gas for this, which prevents a complete DDoS attack on the contract), but such abnormal requests could be interpreted as “spam” by a dApp and not display them in the

same display units as more legitimate requests (e.g. in a "spam" tab). Another filtering would be that execute requests for unknown addresses are put by the dApp in this "spam" folder. Tricking the identity owner to approve an execution request is an attack vector that does not only apply to native tokens stored on the identity contract, but also to key managements, claim addition/removal and by extension all operations on the Identity Contract and contracts the identity could interact with.

We could mitigate slightly the risk of approving such a trick execution request by requiring the approve method to receive all parameters of the execution request (id, value, to, and data), so that whenever an identity owner signs an approval transaction they could verify the data and the called contract/recipient address by themselves.

We could also have some sort of list of allowed addresses to call execute with certain rules. For instance, only approved addresses could transfer value or interact with keys management methods on the identity. This would mitigate the risk for the identity owned native currency and identity keys, but it won't prevent transactions to other contracts without value (and thus transferring non-permissioned tokens)."

We accept the explanation.)

■■■ High

H01. Denial of Service

In an edge case scenario, an address with a Key of purpose 2 can run the approve function with input _id >= _executionNonce and _approve = true. The code will execute with success and _executions[_id].executed = true will be set.

When the execute() function hits the _executionNonce, it will have status executed and will revert. Blocking any access to execute().

The require(!_executions[_executionNonce].executed, "Already executed") check should never be in the execute() function, but rather in approve() together with additional check that prevents usage of _id > _executionNonce

Path:
./contracts/Identity.sol: execute(), approve()

Recommendation: Re-examine the flow and interactions of the execute() and approve() functions, and rewrite the logic to prevent the possibility of a Denial of Service scenario.

Status: Fixed (Revised commit: 5c40bc7)

H02. Requirements Violation

The implementation of the approve() function in the Identity.sol contract violates the NatSpec specification from the IERC734.sol file.

Triggers Event: `Approved`, `Executed`

This SHOULD require n of m approvals of keys purpose 1, if the _to of the execution is the identity contract itself, to successfully approve an execution.

Violation of requirements can lead to incorrect assumptions about the functionality of the contract.

Paths:
./contracts/interface/IERC734.sol : ExecutionFailed, approve()
./contracts/Identity.sol: approve()

Recommendation: Update the documentation or implement functionalities.

Status: Fixed (Revised commit: c20c148)

H03. Requirements Violation

In the IERC734 interface file, functions are described and events are presented that are not implemented or used in the Identity contract implementation.

Emitted when the list of required keys to perform an action was updated.

event KeysRequiredChanged(uint256 purpose, uint256 number);

Violation of requirements can lead to incorrect assumptions about the functionality of the contract.

Path:
./contracts/interface/IERC734.sol : KeysRequiredChanged

Recommendation: Update the documentation or implement functionalities.

Status: Fixed (Revised commit: 5c40bc7)

H04. Requirements Violation

In the IERC735 interface file, a functionality for claim requests is described, along with the event declaration *ClaimRequested*.

There is no such functionality, nor is the ClaimRequested ever used.

Violation of requirements can lead to incorrect assumptions about the functionality of the contract.

Path:
./contracts/interface/IERC735.sol : ClaimRequested

Recommendation: Update the documentation or implement functionalities.

Status: Fixed (Revised commit: c20c148)

■■ Medium

M01. Inconsistent Data - Unused Return Value

In Identity.execute(), calls to *approve()* are made, but the return value of the boolean is not checked or propagated to the caller.

In the case of an immediate execution of the call, if the msg.sender is a Level 1 or Level 2 key, the return information from the approve() function is lost and can only be accessed by reading the emitted event off-chain.

Path:
./contracts/Identity.sol: execute(), approve()

Recommendation: Consider adding a check for the return value of calls to approve() and propagating that return value within the execute() function.

Status: Mitigated (with Customer notice stated as:

"We don't consider useful to use the return boolean value of approve in the execute function, from our perspective it doesn't add anything of interest to the function as the execution success or failure is handled by the approve function itself and events are emitted in both scenarios.")

M02. Contradiction - Missing Validation

In the approve() function, any executed 'Execution' can be replayed without validation to prevent it.

This can lead to unexpected values being processed by the contract.

Path:
./contracts/Identity.sol: approve()

Recommendation: Implement validation to prevent the execution of already executed 'Executions'.

Status: Fixed (Revised commit: c20c148)

M03. Best Practice Violation - Checks-Effects-Interactions Pattern

In the Identity.sol contract, during the function executions, some state variables are updated after the external calls, which is against best practices.

This may lead to confusion, reentrancies, race conditions, and denial of service vulnerabilities during the implementation of new functionality.

- In the *approve()* function of the Identity.sol contract, *_executions[_id].to.call* call is made before updating *_executions[_id].executed = true*.
- In the *execute()* function of the Identity.sol contract, *approve()* external call is made before doing state changes on _executionNonce.

Path:
./contracts/Identity.sol : approve(), execute()

Recommendation: Common best practices should be followed, functions should be implemented according to the Checks-Effects-Interactions pattern. If not possible, the nonReentrant modifier can be used.

- In *approve()* of Identity.sol, *_executions[_id].to.call* should not be made before updating *_executions[_id].executed = true*.
- In the *execute()* function of the Identity.sol contract, *approve()* should be called after doing state changes on _executionNonce.

Status: Fixed (Revised commit: 5c40bc7)

M04. Unscalable Functionality - Code Duplication

In Identity.sol and ClaimIssuer.sol, several functions have the following check, instead of using a modifier:

```
if (msg.sender != address(this)) {
      require(keyHasPurpose(keccak256(abi.encode(msg.sender)),1),
"Permissions: Sender does not have management key");
}
```

Duplicated logic takes more Gas for deployment and makes further development difficult. A modifier should be used, with an argument for the key number that will be checked.

Paths:
./contracts/Identity.sol: addKey(), removeKey(), addClaim(), removeClaim()
./contracts/ClaimIssuer.sol: revokeClaim()

Recommendation: Consider using a modifier for these checks.

Status: Fixed (Revised commit: 5c40bc7)

M05. Inefficient Gas Model - Storage Abuse

In several cases, storage variables are read/written many times, although memory variables can be used.

In Identity.addKey(), _keys[_key].purposes is computed twice at every loop, to check the length and to compute purpose. Instead of this, the purposes array should be saved as a memory variable and used inside the loop.

Path:
./contracts/Identity.sol: addKey()

Recommendation: Use memory variables instead of storage when repeating operations.

Status: Fixed (Revised commit: 5c40bc7)

M06. Inefficient Gas Model - Storage Abuse

In several cases, storage variables are read/written many times, although memory variables can be used.

In Identity.execute(), storage variable _executionNonce is read repeatedly. Instead, create a new variable (e.g. execNonce) and use it over the whole function. Update the storage variable _executionNonce at the end of the function.

Path:
./contracts/Identity.sol: execute()

Recommendation: Use memory variables instead of storage when repeating operations.

Status: Fixed (Revised commit: 5c40bc7)

M07. Inefficient Gas Model - Storage Abuse

In several cases, storage variables are read/written many times, although memory variables can be used.

In Identity.removeKey(), the storage variables _keys[_key].purposes and _keysByPurpose[_purpose] are used repeatedly. Use a memory variable instead.

Path:
./contracts/Identity.sol: removeKey()

Recommendation: Use memory variables instead of storage when repeating operations.

Status: Fixed (Revised commit: c20c148)

M08. Inefficient Gas Model - Storage Abuse

In several cases, storage variables are read/written many times, although memory variables can be used.

In Identity.removeClaim(), the storage variable _claims[_claimId].topic is accessed many times. Use a memory variable instead.

Path:
./contracts/Identity.sol: removeClaim()

Recommendation: Use memory variables instead of storage when repeating operations.

Status: Fixed (Revised commit: 5c40bc7)

M09. Best Practice Violation - Missing Validation

In Identity.removeKey(), a while loop checks whether _keysByPurpose[_purpose][keyIndex] != _key. Consider adding a check so that the loop stops if the whole array is checked, as it is done in while (_keys[_key].purposes[purposeIndex] != _purpose).

Path:
./contracts/Identity.sol: removeKey()

Recommendation: Add a check to make sure the loop stops at the end of the array.

Status: Fixed (Revised commit: 5c40bc7)

M10. Best Practice Violation - Missing Validation

In Identity.removeClaim(), a while loop checks whether _claimsByTopic[_claims[_claimId].topic][claimIndex] != _claimId. Consider adding a check so that the loop stops if the whole array is checked, as it is done in while (_keys[_key].purposes[purposeIndex] != _purpose).

Path:
./contracts/Identity.sol: removeClaim()

Recommendation: Add a check to make sure the loop stops at the end of the array.

Status: Fixed (Revised commit: 5c40bc7)

M11. Unscalable Functionality

In Identity.sol.constructor(), the state variable _canInteract in Storage.sol should be set to false by default (other functions must adapt).

Going back to Identity.sol.constructor(): _canInteract = !_isLibrary is unnecessary and the condition statement can be changed to if(!_isLibrary).

When __Identity_init() is called, the state variable _canInteract will be set to true.

Path:
./contracts/proxy/ImplementationAuthority.sol : updateImplementation()

Recommendation: Update the code to an optimized structure.

Status: Fixed (Revised commit: 5c40bc7)

M12. Contradiction - NatSpec Comment Contradiction

In the Identity.sol contracts *addKey()* function, documentation states that the uint256 _purpose parameter is actually a type uint256[].

Path:
./contracts/Identity.sol : addKey()

Recommendation: Either correct documentation or the code.

Status: Fixed (Revised commit: 5c40bc7)

M13. Best Practice Violation - Lock of Native Tokens

In the execute() function, native tokens are accepted as part of the call. If the 'Execution' is rejected during approval, the funds will not be used and will be locked inside the contract.

Path:
./contracts/Identity.sol: approve()

Recommendation: Consider implementing functionality that provides refunds in case of rejection.

Status: Mitigated (with Customer notice stated as:

“Tokens would still be accessible because a subsequent Execute Request could be emitted to transfer those native tokens to another address (including the original sender address). The possibility to refund native currency to the original sender should be discussed. (As of now, we don’t have a method to permanently decline an Execution Request). We could add a `execution.rejected` property, set to true whenever an execution is explicitly rejected, and prevent any other call to approve for this execution.”

We agree that in the current form of the contracts, this statement is correct.)

M14. Inconsistent Data - Incorrect Event Emitting Order

The order of events triggered during the execution of the *execute()* and *approve()* functions is incorrect.

In a situation where the sender of the *execute()* function is an *ACTION* or *MANAGEMENT* key, the only event that should be triggered is *Executed*.

The *ExecutionRequested* event should only be triggered in situations when the sender of the *execute()* function is not an *ACTION* or *MANAGEMENT* key, in order to inform the off-chain system about pending executions.

The *Approved* event should only be triggered when the *approve()* function is called directly by an account with an *ACTION* or *MANAGEMENT* key, and not from the *execute()* function during the instant execution flow. The *Executed* event can be triggered after the *Approved* in a situation when execution was approved and it was a successful execution.

The *ExecutionFailed* event will never be emitted, as the transaction will always revert in the block of code where the event is triggered, making this event redundant.

Path:
./contracts/Identity.sol: : execute(), approve()

Recommendation: Re-examine how the events should be triggered for each of the execute() and approve() functions' possible flows.

Status: Fixed (Revised commit: c20c148)

M15. Inefficient Gas Model - Storage Abuse

In the Identity.sol contracts *removeClaim()* function, the storage variable _climsByTopic[_topic].length is being accessed multiple times in the following cases:

```
while (_claimsByTopic[_topic][claimIndex] != _claimId) {
      claimIndex++;

      if (claimIndex >= _claimsByTopic[_topic].length) {
            break;
      }
}

_claimsByTopic[_topic][claimIndex] =
_claimsByTopic[_topic][_claimsByTopic[_topic].length - 1];
```

Assigning memory variables to read-only operations would save gas and increase readability.

Path: ./contracts/Identity.sol : removeClaim()

Recommendation: Use memory variables instead of storage when repeating operations.

Status: Fixed (Revised commit: c20c148)

Low

L01. Unused Import

In IdFactory.sol, the file IImplementationAuthority.sol is imported but never used. Unused imports should be removed from the contracts. Unused imports are allowed in Solidity and do not pose a direct security issue. It is best practice to avoid them as they can decrease readability.

Path:
./contracts/factory/IdFactory.sol

Recommendation: Remove unused imports.

Status: Fixed (Revised commit: 5c40bc7)

L02. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of *0x0*. This can lead to unwanted external calls to *0x0*.

Paths:
./contracts/IdFactory.sol : constructor(), __Identity_init()
./contracts/proxy/IdentityProxy.sol : constructor()
./contracts/proxy/ImplementationAuthority.sol : constructor(), updateImplementation()

Recommendation: Add zero address check.

Status: Fixed (Revised commit: 5c40bc7)

L03. Redundant Check

In IdFactory.removeTokenFactory(), zero address is checked with require(_factory != address(0), "invalid argument - zero address"). However, this check is unnecessary since the next requirement is checking if the address is a Factory, which will exclude 0x0.

In Identity.removeKey(), there is a check for _keys[_key].purposes.length > 0. However, this requirement can be removed since it is already contained in _keys[_key].key == _key, because all keys are added with at least one purpose.

Paths:
./contracts/factory/IdFactory.sol : removeTokenFactory()
./contracts/Identity.sol : removeKey()

Recommendation: Remove redundant checks.

Status: Fixed (Revised commit: 5c40bc7)

L04. Unspecific Storage

In IdFactory.createTokenIdentity(), new token identities are linked to token addresses but stored as regular wallets in *_userIdentity[_token] = identity* and *_wallets[identity].push(_token)*. Using different storage variables is recommended to keep things specific.

Path:
./contracts/factory/IdFactory.sol: createTokenIdentity()

Recommendation: Add specific storage variables for wallets or tokens.

Status: Fixed (Revised commit: 5c40bc7)

L05. Unclear Error Message

In IdFactory.linkWallet(), the error message in require(_wallets[identity].length <= 100, "not more than 100 _wallets linked") is not clear. Add the word “allowed” or similar to make it better.

Path:
./contracts/factory/IdFactory.sol : linkWallet()

Recommendation: Rewrite the error message.

Status: Fixed (Revised commit: 5c40bc7)

L06. Gas Optimization in Comparison

In IdFactory.linkWallet(), in require(_wallets[identity].length <= 100, "not more than 100 _wallets linked"), the Gas cost can be decreased by using <101 instead.

In Identity.removeKey(), the condition if(purposeIndex >= _keys[_key].purposes.length) can be optimized to if(purposeIndex = _keys[_key].purposes.length). In addition, when the condition is met, instead of a break and the following require(purposeIndex < _keys[_key].purposes.length), a revert with the error message "NonExisting: Key doesn't have such purpose" can save some extra Gas.

Path:
./contracts/factory/IdFactory.sol: linkWallet(), removeKey()

Recommendation: Update to the cheaper version of code.

Status: Fixed (Revised commit: 5c40bc7)

L07. Style Guide Violation

In IdFactory and IIdFactory, there is a blank space before the brackets in *isTokenFactory ()*, *addTokenFactory()* and *removeTokenFactory()* functions.

Paths:
./contracts/IdFactory.sol: isTokenFactory(), addTokenFactory(), removeTokenFactory()
./contracts/IIdFactory.sol: isTokenFactory(), addTokenFactory(), removeTokenFactory()

Recommendation: Remove the blank space.

Status: Fixed (Revised commit: 5c40bc7)

L08. Naming Consistency

The naming of variables does not follow consistency. In some cases, _mixedCase is used, while others are mixedCase. This can lead to confusion since the different naming leads to assuming different properties.

All state variables are named in _mixedCase, with the exception of *revokedClaims* in ClaimIssuer.sol.

Input variables in different functions use _mixedCase or mixedCase arbitrarily.

Path:
./contracts/ClaimIssuer.sol: revokedClaims

Recommendation: Use the same naming pattern for all variables or document why they need to be different.

Status: Mitigated (Internal variables are named in _mixedCase while public variables are named in mixedCase.)

L09. Functions that Can Be Declared External

In order to save Gas, *public* functions that are never called in the contract should be declared as *external*.

Paths:
./contracts/proxy/ImplementationAuthority.sol : updateImplementation()
./contracts/Identity.sol : initialize(), execute(), getClaimIdsByTopic(), getKey(), getKeyPurposes(), getKeyByPurpose()
./contracts/version/Version.sol : version()

Recommendation: Use the *external* attribute for functions that are never called from the contract.

Status: Fixed (Revised commit: 5c40bc7)

L10. Style Guide - Redundant Code

In the Identity.sol contract's *addClaim()* function, there are two big blocks of code for _claims[claimId] that are repeated in the conditional if/else statements unnecessarily. This leads to higher Gas cost and decreases readability.

Path:
./contracts/Identity.sol : addClaim()

Recommendation: Rewrite the if/else statement so that code is not duplicated.

Status: Fixed
(Revised commit: c20c1480b5e2ad10d3ffd23d9d760f69f1645825)

L11. Best Practice Violation - Shadowing State Variable

In the IdFactory.sol contracts *createTokenIdentity()* function, the _owner variable shadows' existing variable Ownable._owner.

This may lead to undesired behavior.

Path:
./contracts/factory/IdFactory.sol : createTokenIdentity()

Recommendation: Consider using a different naming for the variable.

Status: Fixed (Revised commit: 5c40bc7)

L12. Unfinished NatSpec

NatSpec is not complete - some Smart Contract members are undocumented.

Paths:
./contracts/Identity.sol
./contracts/factory/IdFactory.sol
./contracts/interface/IERC734.sol
./contracts/proxy/IdentityProxy.sol

Recommendation: Add NatSpec to undocumented members of the Smart Contracts.

Status: Fixed (Revised commit: 5c40bc7)

L13. State Variables that Could Be Declared as Immutable

There are variables in the contract that can be declared as immutable to save Gas.

Path:
./contracts/factory/IdFactory.sol : _implementationAuthority

Recommendation: Variables that do not change after construction execution should be declared as immutable.

Status: Fixed (Revised commit: 5c40bc7)

L14. Redundant Variable

In the Storage.sol contract, there is a variable declaration named *_identifier* which is not used anywhere.

Redundant code might decrease code readability and increase gas consumption.

Path:
./contracts/storage/Storage.sol : _identifier

Recommendation: Remove redundant code.

Status: Fixed (Revised commit: c20c148)

L15. Redundant Require Statement

In the Identity.sol contracts *execute()* function, there is a requirement check:

require(!_executions[_executionId].executed, "Already executed");

This is redundant since the *_executionNonce* is being incremented after every *execute()* call and there is no possibility for a previous *_executionId* to be processed when *execute()* call is made.

Path:
./contracts/Identity.sol : execute()

Recommendation: Remove redundant require statement to increase readability and Gas efficiency.

Status: Fixed (Revised commit: c20c148)

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.