HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Abacus
Date: September 19th, 2022

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 Abacus
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Interchain message service
Platform	EVM
Network	Ethereum, BSC
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.useabacus.network/
Timeline	01.08.2022 - 19.09.2022
Changelog	12.08.2022 - Initial Review 19.09.2022 - Second Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Abacus (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/abacus-network/abacus-monorepo/tree/audit-scope-0
Commit:
24dc33c976cc51364abc5f5a63cf5ce46b84bbf1
Technical Documentation:
Whitepaper (partial functional requirements provided)

General repository instructions

Technical description

Integration and Unit Tests: Yes
Contracts:
File: ./solidity/core/contracts/libs/Merkle.sol
SHA3: 26bda8009ee094c60b3c0c10ec454bc3bfffbc74d45c49fd1ed50d696aed7751

File: ./solidity/core/contracts/libs/Message.sol
SHA3: 2ead151cdd7d8193950405628e5a20e991109bc81262a5f7b0ce280a8af463c1

File: ./solidity/core/contracts/libs/TypeCasts.sol
SHA3: a6aba3f1569a897fef473a4a05d632db8b111eb41fe48f8c36c27a7d382a028a

File: ./solidity/core/contracts/upgrade/UpgradeBeacon.sol
SHA3: 48b14dde31005bf4426dc0c4bd65e87379348500c0bf4f947ba9bc04ca168564

File: ./solidity/core/contracts/upgrade/UpgradeBeaconController.sol
SHA3: 8ea1331d0e07ddb67bd6d52e53522b62aeea380b918219332e40964945cdbe39

File: ./solidity/core/contracts/upgrade/UpgradeBeaconProxy.sol
SHA3: 7f96ee4188cffa4d478d541eeaae5fdb1a403e409b4ef0b2ac02d69201dffd6c

File: ./solidity/core/contracts/validator-manager/InboxValidatorManager.sol
SHA3: 4189c2eee44795ae1e6b24376f3030c13bfdee0ff816bb6a33138b0ce8de8682

File: ./solidity/core/contracts/validator-manager/
MultisigValidatorManager.sol
SHA3: 160f595417b01479062dbf15566898b199ac3ea701ca3ebf06afb07c47563d3c

File: ./solidity/core/contracts/validator-manager/
OutboxValidatorManager.sol
SHA3: 4e45c04b1665fe1ba9b5ddc2ce2fa9be06acbd4c43445bad8d4fa8950ced2099

File: ./solidity/core/contracts/AbacusConnectionManager.sol
SHA3: cb07d82574faea7a7286685f50eecb4a7f815fa382d4c850abdeb4913bb5b080

File: ./solidity/core/contracts/Inbox.sol
SHA3: 15773ac4d11dcda43910c291b213da81ef3e62009df57ede81d9d82f3b15a444

File: ./solidity/core/contracts/InterchainGasPaymaster.sol
SHA3: 1c930d51238016674d986ecb5e6816371bc1cadf1c58d2eaffdb6ff71e69e83c

File: ./solidity/core/contracts/Mailbox.sol
SHA3: 29e6f9d11cf1c40a66d0453938a6bd8657381beedaf950483967f55c831e6514

File: ./solidity/core/contracts/MerkleTreeManager.sol
SHA3: 69a11aaee626c1546fefce4070f3ae9691ed2b4ada452455d3a945b546707cfa

File: ./solidity/core/contracts/Outbox.sol
SHA3: 67d53fef05a998291188d9a3b643254ca081d3ca4bb44c671cabfb3873b9364f

File: ./solidity/core/contracts/Version0.sol
SHA3: 4e16eb24d2e65e52b7c568d45177288f98b4db4b49c2da17824851f12ed190ca

File: ./solidity/core/interfaces/IAbacusConnectionManager.sol
SHA3: a8aea1f2c0bf3992edbcda5ec6b6b45b465647307873c91b964781b18a1dc8f0

File: ./solidity/core/interfaces/IInbox.sol
SHA3: f22aa00f106c1b553f871b92d98948726342acc9365e9c66597e5f4935de34cf

File: ./solidity/core/interfaces/IInterchainGasPaymaster.sol
SHA3: 51a202396b7d72003b8e4f4fa595fb1341b1267e5df5c2204e62b8710d425622

File: ./solidity/core/interfaces/IMailbox.sol
SHA3: fb450d381651a13bcfe25ed43693b824f19b4b3f71b0ead552ccee89cc282344

File: ./solidity/core/interfaces/IMessageRecipient.sol
SHA3: df770d6ff439c2300c4ba039ecdc5eb5757db06d8df98d64a47659204c4bfd1b

File: ./solidity/core/interfaces/IOutbox.sol
SHA3: 34e28426db9f6e9406e75fe24648b2043cdd29190e897b9a96c61acb4bb9279b

Second review scope

Repository:
https://github.com/abacus-network/abacus-monorepo/tree/hacken-fixes
Commit:
bbf3abfe0588a0842fcec0e7674d08b0227e9a43
Technical Documentation:
Whitepaper (partial functional requirements provided)

General repository instructions

Technical description

Integration and Unit Tests: Yes
Contracts:
File: ./solidity/core/contracts/libs/Merkle.sol
SHA3: 26bda8009ee094c60b3c0c10ec454bc3bfffbc74d45c49fd1ed50d696aed7751

File: ./solidity/core/contracts/libs/Message.sol
SHA3: 2ead151cdd7d8193950405628e5a20e991109bc81262a5f7b0ce280a8af463c1

File: ./solidity/core/contracts/libs/TypeCasts.sol
SHA3: a6aba3f1569a897fef473a4a05d632db8b111eb41fe48f8c36c27a7d382a028a

File: ./solidity/core/contracts/upgrade/UpgradeBeacon.sol
SHA3: 48b14dde31005bf4426dc0c4bd65e87379348500c0bf4f947ba9bc04ca168564

File: ./solidity/core/contracts/upgrade/UpgradeBeaconController.sol
SHA3: 8ea1331d0e07ddb67bd6d52e53522b62aeea380b918219332e40964945cdbe39

File: ./solidity/core/contracts/upgrade/UpgradeBeaconProxy.sol
SHA3: 7f96ee4188cffa4d478d541eeaae5fdb1a403e409b4ef0b2ac02d69201dffd6c

File: ./solidity/core/contracts/validator-manager/InboxValidatorManager.sol
SHA3: 113cfc400104ad83cc5ab1c2672c0cac5ff4accb7e5e0a190a0d9d1b7b9080f9

File: ./solidity/core/contracts/validator-manager/
MultisigValidatorManager.sol
SHA3: 7d7005fd0c8cca9a79a9aabe4b9aff40da91bce88d16bd62bfc2f0893387c9d7

File: ./solidity/core/contracts/validator-manager/
OutboxValidatorManager.sol
SHA3: f0b0f0531b0a775aa29c20ba5de1a8b1651a04af1ceebf72e3eef975289ee40f

File: ./solidity/core/contracts/AbacusConnectionManager.sol
SHA3: 27cb1a87868564ecf7ce1a051a1c2b6ba72b814358b9717431535e005d42b1e2

File: ./solidity/core/contracts/Inbox.sol
SHA3: 00d362bb88d0432029ad5b6d7999479ac50614ff7f649f7b36d7f40af96cafa7

File: ./solidity/core/contracts/InterchainGasPaymaster.sol
SHA3: 1c930d51238016674d986ecb5e6816371bc1cadf1c58d2eaffdb6ff71e69e83c

File: ./solidity/core/contracts/Mailbox.sol
SHA3: 29e6f9d11cf1c40a66d0453938a6bd8657381beedaf950483967f55c831e6514

File: ./solidity/core/contracts/MerkleTreeManager.sol
SHA3: 69a11aaee626c1546fefce4070f3ae9691ed2b4ada452455d3a945b546707cfa

File: ./solidity/core/contracts/Outbox.sol
SHA3: 3e07b2d4e87a65f16d140229239d742abc75e2fec4de61d6d2b6bdaf474b68ea

File: ./solidity/core/contracts/Versioned.sol
SHA3: 7b1e8357094ec9676a04cf8b6b6e022155c12f9949688dd9a6479e5640716a60

File: ./solidity/core/interfaces/IAbacusConnectionManager.sol
SHA3: a8aea1f2c0bf3992edbcda5ec6b6b45b465647307873c91b964781b18a1dc8f0

File: ./solidity/core/interfaces/IInbox.sol
SHA3: f22aa00f106c1b553f871b92d98948726342acc9365e9c66597e5f4935de34cf

File: ./solidity/core/interfaces/IInterchainGasPaymaster.sol
SHA3: 51a202396b7d72003b8e4f4fa595fb1341b1267e5df5c2204e62b8710d425622

File: ./solidity/core/interfaces/IMailbox.sol
SHA3: b058c5fb24b6112fb4f992dfc80ada083c60317ee726c39b1a89d733528d7108

File: ./solidity/core/interfaces/IMessageRecipient.sol
SHA3: df770d6ff439c2300c4ba039ecdc5eb5757db06d8df98d64a47659204c4bfd1b

```
File: ./solidity/core/interfaces/IMultisigValidatorManager.sol
SHA3: 8dac47c1aa7de2dfe66381a2cdb6cadb4c8bb08d070fbab97a7e1e0149fa52c4

File: ./solidity/core/interfaces/IOutbox.sol
SHA3: 34e28426db9f6e9406e75fe24648b2043cdd29190e897b9a96c61acb4bb9279b
```

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

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.

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

Executive Summary

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

Documentation quality

The total Documentation Quality score is **6** out of **10**. An overview of the whole system is provided, but detailed functional requirements are missed. The technical description is clear.

Code quality

The total CodeQuality score is **7** out of **10**. The code follows official language style guides and is mostly covered with unit tests, some negative cases are not covered. **Test coverage is 81%**. The library code has some unused functions. Code duplications found.

Architecture quality

The architecture quality score is **7** out of **10**. Smart contracts of the project follow the best practices. The project has a well-configured development environment. Only general architecture is documented, the mission of some contracts is not obvious.

Security score

As a result of the audit, the code contains **1** high, **1** medium, **5** low severity issues. The security score is **4** 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: **4.8**.

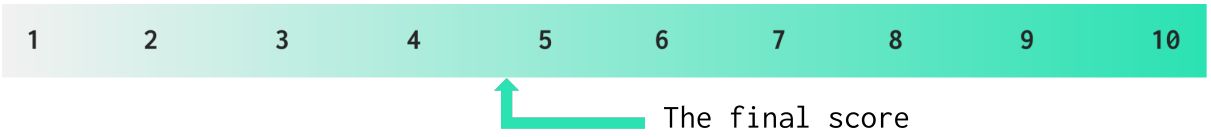

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 August 2022	7	5	2	1
19 September 2022	5	1	1	0

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

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.	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 it is 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.	Passed
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 identifier should always be used. All parameters from the signature should be used in signer recovery	Failed
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, that may be changed in the future.	Passed

System Overview

Abacus is an interchain message system. Its purpose is to allow users to build interchain decentralized applications. The scope of the audit contains such contracts used for messaging to/from EVM compatible networks:

- *Versioned* - inheritable contract that provides *VERSION* public constant
- *Mailbox* - inheritable contract that provides *onlyValidatorManager* modifier and concomitant logic
- *MerkleTreeManager* - inheritable contract that provides variable *tree* and function *root* that returns root of the tree
- *InterchainGasPaymaster* - contract for storing Ether as a fee for the system usage
- *Outbox* - contract for receiving messages, may be failed if corrupted data was signed by validators
- *Inbox* - contract that sends validated messages to receivers
- *AbacusConnectionManager* - contract that manages outbox and inbox contracts in the chain of deployment
- *UpgradeBeacon contracts* - custom implementation of the upgradable beacon pattern
- *ValidatorManager contracts* - multisignature contracts, which control message processing power of the system

Privileged roles

- The owner of *UpgradeBeaconController* can upgrade contract code of:
  - *Inbox*, *Outbox*
  - *MerkleTreeManager*
  - *InterchainGasPaymaster*
- The owner of *Inbox*, *Outbox* contracts can update *ValidatorManager contracts* for them
- The owner of *InterchainGasPaymaster* contract obtains all Ether sent to the contract
- The owner of *ValidatorManager contracts* can update validators list

Risks

- The contracts in the system are upgradable; this allows the administrator to update the contract logic.
- The administrator of the contract may add new validators, which makes the system dependent on the owner's behavior.
- The majority of Validators may send any arbitrary messages. Administrators may increase the possibility of signing an arbitrary message by setting a low threshold value, which allows the minority of validators to make the decisions.

- The *Outbox* contract may be in Failed state, this leads to inability to process new messages.
- In case of domain collision, messages may be delivered to the wrong chain, and the *Outbox* contracts of the collision chains may fail regularly.
- There is no guarantee that any messages will be delivered, but it is guaranteed that if message X is signed by a validator, it is possible to deliver messages up to and including the X.
- There is no guarantee of receiving fraudulent messages, but according to the documentation, there are watchtowers which should stop the system in case of fraudulence.
- On the destination side, the recipient's contract “handle” function may be called by any caller with arbitrary data; this requires checking if the sender is an expected Inbox contract.

Findings

■■■■ Critical

1. Signed message replay attack

As it is possible to set several inboxes per domain in *AbacusConnectionManager* and *Inbox* contracts only check if the message was not processed in the contract before, it is possible to execute one message more than once.

Note: changing *Inbox* contract may lead to the same consequences as the list of processed messages is clear on creation.

This may lead to the users' funds loss.

Path: ./solidity/core/contracts/AbacusConnectionManager.sol : domainToInboxes

Recommendation: provide bijective relation from domain to inbox and remove the possibility of changing inbox contracts.

Status: Fixed (Revised commit: bbf3abfe0588a0842fcec0e7674d08b0227e9a43)

■■■ High

1. Denial of Service vulnerability

The system has the *Outbox* contract, which may be set to *Failed* state by the *fail* function call if a quorum of validators signed a non-existent message. The message signed by the quorum may be used more than once to make the contract *Failed*.

This may lead to blocking the *Outbox* contract functionality repeatedly.

Path:
./solidity/core/contracts/validator-manager/OutboxValidatorManager.sol : prematureCheckpoint(), fraudulentCheckpoint()

Recommendation: rework the logic to deny the double usage of the signed message.

Status: Reported (The impact of *fail()* ing the *Outbox* multiple times is trivial because once it is failed, there is no mechanism to recover. However, we acknowledge the need to prevent evidence reuse once economic slashing is implemented and are tracking this work in the issue)

2. Undocumented behaviour

There is an *InterchainGasPaymaster* contract implemented, which aims to take fees when a user sends a message using the system. However,

it is possible to avoid paying fees if calling the *dispatch* function in the *Outbox* contract directly.

This may lead to no fees paid to validators for their work.

Path: ./solidity/core/contracts/Outbox.sol : dispatch()

Recommendation: add the fees related logic to the *Outbox* contract to prevent free *dispatch* function calls.

Status: Mitigated (There is currently no fee implemented to pay validators for their work on individual messages. The InterchainGasPaymaster functionality is documented here: https://docs.useabacus.network/abacus-docs/developers/messaging-api/gas)

Medium

1. Unwanted transaction reverting

As the *Outbox* contract may be recreated after failing, the first transactions on the new instance may be the same as in the old instance. In such a situation, the same root of the Merkle tree appears, and messages may not be considered for executing in the Inbox contract.

This may lead to unwanted transactions reverting.

Path: ./solidity/core/contracts/Outbox.sol : dispatch()

Recommendation: send a specific zero system message with salt to prevent the case.

Status: Mitigated (Each deployment is versioned such that new deployments are completely independent from previous deployments. Because the replay protection provided by *Inboxes* is distinct for each *Outbox* , there is no danger that identical messages sent to multiple *Outboxes* will cause messages to be dropped)

2. Requirement violation

According to the documentation, watchtowers should be able to check signed data for fraudulence. However, it is impossible to check the first message until the next message is sent. It happens so, as it is not possible to run *cacheCheckpoint* function when the count of messages is 1.

Note: *cachedCheckpoints* function returns 0 for all non-existent roots, so removing zero checks is not applicable.

Paths:
./solidity/core/contracts/Outbox.sol : latestCheckpoint(), cacheCheckpoint()
./solidity/core/contracts/validator-manager/OutboxValidatorManager.sol : fraudulentCheckpoint()

Recommendation: send a specific zero system message to fill the space out or reimplement leaf indexing from 1.

Status: Mitigated (In this case, the watchtower can atomically dispatch another message and then prove fraud in the same transaction)

3. **Missing check for return value**

Return value should be taken into account.

Path: ./solidity/core/contracts/AbacusConnectionManager.sol : _unenrollInbox(), _enrollInbox()

Recommendation: check return values in all cases.

Status: Fixed (Revised commit: bbf3abfe0588a0842fcec0e7674d08b0227e9a43)

4. **Code duplication**

The *Message* library has the *leaf* function, which calculates the hash for message and leaf index data, but in the *Outbox* contract in the *dispatch* function, the hash *_messageHash* for the message and leaf index is calculated by the duplicated code.

Code duplication leads to unneeded Gas usage during the contract deployment.

Path: ./solidity/core/contracts/Outbox.sol : dispatch()

Recommendation: use *leaf* library function for the hash calculations.

Status: Mitigated (It is not possible to reuse the leaf function from the Message library because of the calldata and memory type conflict.)

5. **Double event emitting**

Some functions in the contract system may be called more than once, which may redundantly emit the events.

Redundant event emitting may confuse users and has a negative effect on the frontend side.

Paths:

./solidity/core/contracts/Mailbox.sol : _setValidatorManager()
./solidity/core/contracts/validator-manager/OutboxValidatorManager.sol : fraudulentCheckpoint(), prematureCheckpoint()
./solidity/core/contracts/Outbox.sol : cacheCheckpoint(), fail()
./solidity/core/contracts/AbacusConnectionManager.sol : _unenrollInbox()

Recommendation: add conditional statements to avoid redundant emitting of the same contracts.

Status: Mitigated (This behavior of potential redundancy in emitted event topics due to no-op storage variable changes is consistent with popular ecosystem contracts)

6. Unoptimized loops usage

The system has contracts with the external functions, which return an array of addresses from the `EnumerableSet.AddressSet`. The functions iterate over the items one by one to return the values from the set.

This may lead to redundant gas usage during the interaction with the contracts.

Paths:
./solidity/core/contracts/AbacusConnectionManager.sol : getInboxes()
./solidity/core/contracts/MultisigValidatorManager.sol : validators()

Recommendation: *EnumerableSet* has native field *_inner._values* with the array of values; consider using the field to avoid iterations over values if the return type of *bytes32[]* is appropriate.

Status: New

Low

1. Missing zero address validation

Address parameters are used without checking against the possibility of being *0x0*.

This can lead to unwanted external calls to *0x0*.

Paths:
./solidity/core/contracts/validator-manager/MultisigValidatorManager.sol : _enrollValidator(), _unenrollValidator()
./solidity/core/contracts/AbacusConnectionManager.sol : setOutbox(), enrollInbox()

Recommendation: implement zero address validations.

Status: Fixed (Revised commit: bbf3abfe0588a0842fcec0e7674d08b0227e9a43)

2. Different versions of Solidity are used

The interfaces and contract implementation use different Solidity versions.

Contracts should use the same compiler version to ensure that the contracts in the system are fully compatible.

Recommendation: use one Solidity version.

Status: Reported

3. **Default visibility**

   The lack of variable visibility may cause unexpected variable visibility in derived contracts.

   Path: ./solidity/core/contracts/AbacusConnectionManager.sol : domainToInboxes

   Recommendation: specify the needed visibility during the variable initialization.

   Status: Fixed (Revised commit: bbf3abfe0588a0842fcec0e7674d08b0227e9a43)

4. **Redundant pragma abicoder statement**

   As per the list of v0.8 breaking changes, the ABIEncoderV2 is not experimental anymore - it is enabled by default by the compiler.

   Path: ./solidity/core/contracts/AbacusConnectionManager.sol

   Recommendation: remove redundant *pragma abiencoder v2* statement.

   Status: Fixed (Revised commit: bbf3abfe0588a0842fcec0e7674d08b0227e9a43)

5. **Interface implementation mismatch**

   Interfaces do not match the implementation.

   Paths:
   ./solidity/core/interfaces/IInbox.sol : remoteDomain()
   ./solidity/core/interfaces/IOutbox.sol : count()

   Recommendation: update interfaces according to implementation.

   Status: Reported

6. **Floating pragma**

   The contracts use floating pragma *>=0.8.0* and *>=0.6.11*.

   Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

   Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

   Status: Reported

7. **Possible Gas limit exceeding**

   The extraction of a significant amount of data may lead to the Gas limit exceeding during the calls interaction with the contract's external functions from the other contracts.

Paths:
./contracts/validator-manager/MultisigValidatorManager.sol : validators()
./contracts/AbacusConnectionManager.sol : getInboxes()

Recommendation: implement page navigation through the data set, or bound the data size.

Status: New

8. **Default visibility**

The lack of variable visibility may cause unexpected visibility in derived contracts.

Path: ./solidity/core/contracts/AbacusConnectionManager.sol : domainHashes

Recommendation: specify the needed visibility during the variable initialization.

Status: New

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.