HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: MetisDAO Foundation
**Date**: December 05th, 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 MetisDAO Foundation |
| **Approved By** | Evgeniy Bezuglyi \| SC Audits Department Head at Hacken OU |
| **Type** | Layer 1 for the Layer 2 Protocol |
| **Platform** | EVM |
| **Network** | Ethereum |
| **Language** | Solidity |
| **Methods** | Manual Review, Automated Review, Architecture Review |
| **Website** | https://bridge.metis.io/home |
| **Timeline** | 20.09.2022 - 05.12.2022 |
| **Changelog** | 17.10.2022 - Initial Review<br>01.11.2022 - Second Review<br>05.12.2022 - Third 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 MetisDAO Foundation (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/MetisProtocol/mvm
**Commit:**
ec2d5e093a6a976ffcf61fbd7e46551241c7155c
**Documentation:**
Functional requirements and technical description
**Integration and Unit Tests:** Yes
**Contracts:**
File:
./packages/contracts/contracts/L1/messaging/IL1CrossDomainMessenger.sol
SHA3: cb53c54698819917f089a50461360d7b6ba1f9d522035331232000e9bb3d28bd

File: ./packages/contracts/contracts/L1/messaging/IL1ERC20Bridge.sol
SHA3: f29c2f3ec5e9739aa813d43211d17e9a37198a2021d82a0f62f14660ddf79050

File: ./packages/contracts/contracts/L1/messaging/IL1StandardBridge.sol
SHA3: 2bca4967ec142a31041cfa19ec5cd71f607d37c1dd5d6c8cb118d7e0207ebd88

File:
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol
SHA3: bd74e7cf911596221f5e07cc564df820857083262969149be9b26d1edb808680

File: ./packages/contracts/contracts/L1/messaging/L1StandardBridge.sol
SHA3: 635bb28b1a840fbf381b74e955e6c5b1cb47e05d0473b1365a94b118bd610896

File:
./packages/contracts/contracts/L1/rollup/ICanonicalTransactionChain.sol
SHA3: 55cb6d47be7b2a19f7fc911f3ffb797f3a1e357fc88f0bcb3ce78757d555abce

File: ./packages/contracts/contracts/L1/rollup/IStateCommitmentChain.sol
SHA3: bc2f80a817c2efb4329c70e8add2c13a2515aef758be0af3703b21591cd7d2fd

File: ./packages/contracts/contracts/L2/messaging/IL2ERC20Bridge.sol
SHA3: 254df419cc2085b551842def494977abf722c143ab15519282ba9155cb836a67

File:
./packages/contracts/contracts/libraries/bridge/CrossDomainEnabled.sol
SHA3: 489191e6f45c755b8527c628d30ac562f0083a26ff658a854e841f2d8ac0a1cb

File:
./packages/contracts/contracts/libraries/bridge/ICrossDomainMessenger.sol
SHA3: ffa00added5539937a4faa2e1036fdfa42d788725a153e58e015b4afff9f0588

File:
./packages/contracts/contracts/libraries/bridge/Lib_CrossDomainUtils.sol

```
      SHA3: 2391eba0f71d257e545250da87609a44cd77e605a5a7c433c3806946d59fe7bd

      File: ./packages/contracts/contracts/libraries/codec/Lib_OVMCodec.sol
      SHA3: 2851087aaca512f6204f0eee51082d938707895f37eb2163a50f23addef32339

      File:
./packages/contracts/contracts/libraries/constants/Lib_DefaultValues.sol
      SHA3: 2f11cec61a5104f6d20bebbfa8cf1bb908592672759bb2466e2c023bac9ab799

      File:
./packages/contracts/contracts/libraries/constants/Lib_PredeployAddresses.sol
      SHA3: cb5bc281908d61ae32a59d36780b6e4d40a9cff30e1efd7abc19ff3226d8e83d

      File:
./packages/contracts/contracts/libraries/resolver/Lib_AddressManager.sol
      SHA3: 52b04a59cbbc88322d12a589f1675172c9199cd0f47f8270ee265fc1400c74fd

      File:
./packages/contracts/contracts/libraries/resolver/Lib_AddressResolver.sol
      SHA3: 59193272d1db167804849a71f84b57b823b07837e957a26f468839d2cd4e97ec

      File: ./packages/contracts/contracts/libraries/rlp/Lib_RLPReader.sol
      SHA3: c919e79452c8db73af0ba57054ce6aa86c2d32bb9c266c5b94bddcc4f3f5567d

      File: ./packages/contracts/contracts/libraries/rlp/Lib_RLPWriter.sol
      SHA3: bf0c9457b5b418eabf60d6e63d675b3fa9ba1dfc9c0bea84770ae6132b96411d

      File: ./packages/contracts/contracts/libraries/trie/Lib_MerkleTrie.sol
      SHA3: 7287bd50865ef7750bf3a9aa9a9b9ed6d0284a0235826df39bf15670e18ba7d9

      File:
./packages/contracts/contracts/libraries/trie/Lib_SecureMerkleTrie.sol
      SHA3: 46294f98bb1fe1c8a5d1cb574ff82dde6235b2668c5b769be76fadabe52416cf

      File: ./packages/contracts/contracts/libraries/utils/Lib_Bytes32Utils.sol
      SHA3: e9f51ba966816c77c57f4ec00ba8788012183560385bf13dcf34a6d15581df1d

      File: ./packages/contracts/contracts/libraries/utils/Lib_BytesUtils.sol
      SHA3: 8a9e0c6bd58aaf7570509428f099801d2795e2368559ffb6e5f6b835c04770db

      File: ./packages/contracts/contracts/standards/AddressAliasHelper.sol
      SHA3: 19d7598fd3946d2c974fcd8080587b1b7a80cf4d1284782bd97bcdd81758ac13

      File: ./packages/contracts/contracts/MVM/iMVM_DiscountOracle.sol
      SHA3: 7852d3fd8cecf1c4b6c368a083f04317c79c9453fbed514528a6f9e9064b7df4

      File: ./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol
      SHA3: ee12a267bd62249696836a9f57c2601bc95ca9bec3021b792a12f62ffeb4e243

      File:
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol
      SHA3: 296e9bee1d0da1061f86984a4ef246de220a6ad6e935d7d0b963a0e6498095b0

      File: ./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol
      SHA3: d0ea3825dc87692219affe8290d576c1311f013397cef982aa039dcf85cdbc25

      File: ./packages/contracts/contracts/libraries/utils/Lib_MerkleTree.sol
      SHA3: 9b9ae7df353c05a78f4d5c5e89c9dde7b5171cbe050435c882010022e3aae189

      File: ./packages/contracts/contracts/L1/verification/IBondManager.sol
```

SHA3: d362a8a02ade76d474c7626b901be78755666cad331f2256815195100f63f76

File: ./packages/contracts/contracts/L1/verification/BondManager.sol
SHA3: 2108f581e70604ff185297dadf966138b3b40e454202b072bae40fb8a391d16b

File: ./packages/contracts/contracts/L1/rollup/IChainStorageContainer.sol
SHA3: 84b56bd49d509b2a8013afce6676c788f6b2d3ed8b9dec0e5ad8722cf95cfd8a

File: ./packages/contracts/contracts/L1/rollup/ChainStorageContainer.sol
SHA3: b086fb41176692e3d0de3147771127874ee22ff631ba1458387d535cf2dec0b1

File: ./packages/contracts/contracts/libraries/utils/Lib_Buffer.sol
SHA3: b518cff5b386c374098e4ae178e7d3a7eaaaf648c6ad874a69d7ba50c107be46

## Second review scope

**Repository:**
https://github.com/MetisProtocol/mvm

**Commit:**
defb4e1dbfa85cf9f8a248ea838b8dacf19860fa

**Documentation:**
Functional requirements and technical description

**Integration and Unit Tests:** Yes

**Contracts:**
File:
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol
SHA3: 1b150a331aac619fc9b2f3151d35339b224c4c09896769d1e161d67049cff17c

File: ./packages/contracts/contracts/L1/messaging/L1StandardBridge.sol
SHA3: a3fbeae3fe7d01cfb83852d4c0ee62b8ccc21285bca414132b656bcaf5568d97

File:
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol
SHA3: 08534d3ec53eddea5051b846b9a17bbe7b98a4ff9050a1af70f8face5faac269

File: ./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol
SHA3: 5a28cdd1cd6fca3beafe3e6cb8fd0dbf78d41cdcdc54955c16d8c465d3c1e6b9

File: ./packages/contracts/contracts/L1/verification/BondManager.sol
SHA3: 4cdb7707e2dd060d3fee5a70aaa5fa8937867d89ed93db92f103d8df480ac8c7

File: ./packages/contracts/contracts/libraries/utils/Lib_Buffer.sol
SHA3: 7fa2362e56f2d81f787476627053572309ffbe427680b5de8f8427b05319aa71

File: ./packages/contracts/contracts/L1/rollup/ChainStorageContainer.sol
SHA3: b086fb41176692e3d0de3147771127874ee22ff631ba1458387d535cf2dec0b1

File: ./packages/contracts/contracts/libraries/utils/Lib_MerkleTree.sol
SHA3: 9b9ae7df353c05a78f4d5c5e89c9dde7b5171cbe050435c882010022e3aae189

File:
./packages/contracts/contracts/L1/messaging/IL1CrossDomainMessenger.sol
SHA3: cb53c54698819917f089a50461360d7b6ba1f9d522035331232000e9bb3d28bd

File: ./packages/contracts/contracts/L1/messaging/IL1ERC20Bridge.sol
SHA3: f29c2f3ec5e9739aa813d43211d17e9a37198a2021d82a0f62f14660ddf79050

File: ./packages/contracts/contracts/L1/messaging/IL1StandardBridge.sol
SHA3: 2bca4967ec142a31041cfa19ec5cd71f607d37c1dd5d6c8cb118d7e0207ebd88

```
      File:
./packages/contracts/contracts/L1/rollup/ICanonicalTransactionChain.sol
      SHA3: 55cb6d47be7b2a19f7fc911f3ffb797f3a1e357fc88f0bcb3ce78757d555abce

      File: ./packages/contracts/contracts/L1/rollup/IStateCommitmentChain.sol
      SHA3: bc2f80a817c2efb4329c70e8add2c13a2515aef758be0af3703b21591cd7d2fd

      File: ./packages/contracts/contracts/L2/messaging/IL2ERC20Bridge.sol
      SHA3: 254df419cc2085b551842def494977abf722c143ab15519282ba9155cb836a67

      File:
./packages/contracts/contracts/libraries/bridge/CrossDomainEnabled.sol
      SHA3: 489191e6f45c755b8527c628d30ac562f0083a26ff658a854e841f2d8ac0a1cb

      File: ./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol
      SHA3: 1b54a1717bfc718fb3673009599c4ff93398134359ebdb0846c3c1623a7cc919

      File:
./packages/contracts/contracts/libraries/bridge/ICrossDomainMessenger.sol
      SHA3: ffa00added5539937a4faa2e1036fdfa42d788725a153e58e015b4afff9f0588

      File:
./packages/contracts/contracts/libraries/bridge/Lib_CrossDomainUtils.sol
      SHA3: 2391eba0f71d257e545250da87609a44cd77e605a5a7c433c3806946d59fe7bd

      File: ./packages/contracts/contracts/libraries/codec/Lib_OVMCodec.sol
      SHA3: 2851087aaca512f6204f0eee51082d938707895f37eb2163a50f23addef32339

      File:
./packages/contracts/contracts/libraries/constants/Lib_DefaultValues.sol
      SHA3: 2f11cec61a5104f6d20bebbfa8cf1bb908592672759bb2466e2c023bac9ab799

      File:
./packages/contracts/contracts/libraries/constants/Lib_PredeployAddresses.sol
      SHA3: cb5bc281908d61ae32a59d36780b6e4d40a9cff30e1efd7abc19ff3226d8e83d

      File:
./packages/contracts/contracts/libraries/resolver/Lib_AddressManager.sol
      SHA3: 52b04a59cbbc88322d12a589f1675172c9199cd0f47f8270ee265fc1400c74fd

      File:
./packages/contracts/contracts/libraries/resolver/Lib_AddressResolver.sol
      SHA3: 59193272d1db167804849a71f84b57b823b07837e957a26f468839d2cd4e97ec

      File: ./packages/contracts/contracts/libraries/rlp/Lib_RLPReader.sol
      SHA3: c919e79452c8db73af0ba57054ce6aa86c2d32bb9c266c5b94bddcc4f3f5567d

      File: ./packages/contracts/contracts/libraries/rlp/Lib_RLPWriter.sol
      SHA3: bf0c9457b5b418eabf60d6e63d675b3fa9ba1dfc9c0bea84770ae6132b96411d

      File: ./packages/contracts/contracts/libraries/trie/Lib_MerkleTrie.sol
      SHA3: 7287bd50865ef7750bf3a9aa9a9b9ed6d0284a0235826df39bf15670e18ba7d9

      File:
./packages/contracts/contracts/libraries/trie/Lib_SecureMerkleTrie.sol
      SHA3: 46294f98bb1fe1c8a5d1cb574ff82dde6235b2668c5b769be76fadabe52416cf


      File: ./packages/contracts/contracts/libraries/utils/Lib_Bytes32Utils.sol
```

SHA3: e9f51ba966816c77c57f4ec00ba8788012183560385bf13dcf34a6d15581df1d

File: ./packages/contracts/contracts/libraries/utils/Lib_BytesUtils.sol
SHA3: 8a9e0c6bd58aaf7570509428f099801d2795e2368559ffb6e5f6b835c04770db

File: ./packages/contracts/contracts/standards/AddressAliasHelper.sol
SHA3: 19d7598fd3946d2c974fcd8080587b1b7a80cf4d1284782bd97bcdd81758ac13

File: ./packages/contracts/contracts/MVM/iMVM_DiscountOracle.sol
SHA3: 61a73a61f52fc298e0d84e8a4cc3b5e4b8a3278cf4cc3fb5f831264024e38264

File: ./packages/contracts/contracts/L1/verification/IBondManager.sol
SHA3: d362a8a02ade76d474c7626b901be78755666cad331f22566815195100f63f76

File: ./packages/contracts/contracts/L1/rollup/IChainStorageContainer.sol
SHA3: 84b56bd49d509b2a8013afce6676c788f6b2d3ed8b9dec0e5ad8722cf95cfd8a

## Third review scope

**Repository:**
https://github.com/MetisProtocol/mvm

**Commit:**
3e388ada0011bf6aefbf0085a8df02ad674cc548

**Documentation:**
Functional requirements and technical description

**Integration and Unit Tests:** Yes

**Contracts:**
File:
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol
SHA3: 0cc99dbbabf101a6bcdf9dbbd6100656fc3e1478589d580c2dafd1c28c3d5a99

File: ./packages/contracts/contracts/L1/messaging/L1StandardBridge.sol
SHA3: 1dc8096e01572c86288af5766a6efca99f7ffa828d722ad6f16c0694eced43c2

File:
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol
SHA3: 40575a61f9fb28c6ac51d63e01e125df80deff726d07b0e9995b3a194d955430

File: ./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol
SHA3: 7b24fa81d454a3999f6e35fc0542957104c421d66c7cd90cfa2fb1db97cc8482

File: ./packages/contracts/contracts/L1/verification/BondManager.sol
SHA3: 41f2ee9ffd15877541d48808893810d04ff382e350e9a6823ffc49dffcb03f18

File: ./packages/contracts/contracts/libraries/utils/Lib_Buffer.sol
SHA3: 7fa2362e56f2d81f787476627053572309ffbe427680b5de8f8427b05319aa71

File: ./packages/contracts/contracts/L1/rollup/ChainStorageContainer.sol
SHA3: fb588de1d4a0f4c618095996dc465669fe21682ceb218fb262a11cfce20626cc

File: ./packages/contracts/contracts/libraries/utils/Lib_MerkleTree.sol
SHA3: 9b9ae7df353c05a78f4d5c5e89c9dde7b5171cbe050435c882010022e3aae189

File:
./packages/contracts/contracts/L1/messaging/IL1CrossDomainMessenger.sol
SHA3: d229208e0304020c7315823d4993a14a2f90031a8aa2bdef8063edd053c116c2

File: ./packages/contracts/contracts/L1/messaging/IL1ERC20Bridge.sol
SHA3: f29c2f3ec5e9739aa813d43211d17e9a37198a2021d82a0f62f14660ddf79050

```
      File: ./packages/contracts/contracts/L1/messaging/IL1StandardBridge.sol
      SHA3: 2bca4967ec142a31041cfa19ec5cd71f607d37c1dd5d6c8cb118d7e0207ebd88

      File:
./packages/contracts/contracts/L1/rollup/ICanonicalTransactionChain.sol
      SHA3: c35f5be5bb8890d5d36ebe7e50530a0e225de528c66a077feacf545f05928755

      File: ./packages/contracts/contracts/L1/rollup/IStateCommitmentChain.sol
      SHA3: 80401c86aac48773746aafd545d60d3f71803e08ef0659fe4215f6b1fa43336c

      File: ./packages/contracts/contracts/L2/messaging/IL2ERC20Bridge.sol
      SHA3: 254df419cc2085b551842def494977abf722c143ab15519282ba9155cb836a67

      File:
./packages/contracts/contracts/libraries/bridge/CrossDomainEnabled.sol
      SHA3: 489191e6f45c755b8527c628d30ac562f0083a26ff658a854e841f2d8ac0a1cb

      File: ./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol
      SHA3: 2febce107c27daccb0eaff2c77fa3ed4a132e1639f73d4d25e1ca45f90b30c56

      File:
./packages/contracts/contracts/libraries/bridge/ICrossDomainMessenger.sol
      SHA3: ffa00added5539937a4faa2e1036fdfa42d788725a153e58e015b4afff9f0588

      File:
./packages/contracts/contracts/libraries/bridge/Lib_CrossDomainUtils.sol
      SHA3: 2391eba0f71d257e545250da87609a44cd77e605a5a7c433c3806946d59fe7bd

      File: ./packages/contracts/contracts/libraries/codec/Lib_OVMCodec.sol
      SHA3: 0f8ccf51663d7c325ec17d8cc5aa2f2219ec5b2ae2f4fec79a1980c5c1dae962

      File:
./packages/contracts/contracts/libraries/constants/Lib_DefaultValues.sol
      SHA3: 2f11cec61a5104f6d20bebbfa8cf1bb908592672759bb2466e2c023bac9ab799

      File:
./packages/contracts/contracts/libraries/constants/Lib_PredeployAddresses.sol
      SHA3: cb5bc281908d61ae32a59d36780b6e4d40a9cff30e1efd7abc19ff3226d8e83d

      File:
./packages/contracts/contracts/libraries/resolver/Lib_AddressManager.sol
      SHA3: 52b04a59cbbc88322d12a589f1675172c9199cd0f47f8270ee265fc1400c74fd

      File:
./packages/contracts/contracts/libraries/resolver/Lib_AddressResolver.sol
      SHA3: 59193272d1db167804849a71f84b57b823b07837e957a26f468839d2cd4e97ec

      File: ./packages/contracts/contracts/libraries/rlp/Lib_RLPReader.sol
      SHA3: c919e79452c8db73af0ba57054ce6aa86c2d32bb9c266c5b94bddcc4f3f5567d

      File: ./packages/contracts/contracts/libraries/rlp/Lib_RLPWriter.sol
      SHA3: bf0c9457b5b418eabf60d6e63d675b3fa9ba1dfc9c0bea84770ae6132b96411d

      File: ./packages/contracts/contracts/libraries/trie/Lib_MerkleTrie.sol
      SHA3: 7287bd50865ef7750bf3a9aa9a9b9ed6d0284a0235826df39bf15670e18ba7d9

      File:
./packages/contracts/contracts/libraries/trie/Lib_SecureMerkleTrie.sol
      SHA3: 46294f98bb1fe1c8a5d1cb574ff82dde6235b2668c5b769be76fadabe52416cf
```

```
File: ./packages/contracts/contracts/libraries/utils/Lib_Bytes32Utils.sol
SHA3: e9f51ba966816c77c57f4ec00ba8788012183560385bf13dcf34a6d15581df1d

File: ./packages/contracts/contracts/libraries/utils/Lib_BytesUtils.sol
SHA3: 8a9e0c6bd58aaf7570509428f099801d2795e2368559ffb6e5f6b835c04770db

File: ./packages/contracts/contracts/standards/AddressAliasHelper.sol
SHA3: 19d7598fd3946d2c974fcd8080587b1b7a80cf4d1284782bd97bcdd81758ac13

File: ./packages/contracts/contracts/MVM/iMVM_DiscountOracle.sol
SHA3: 61a73a61f52fc298e0d84e8a4cc3b5e4b8a3278cf4cc3fb5f831264024e38264

File: ./packages/contracts/contracts/L1/verification/IBondManager.sol
SHA3: d362a8a02ade76d474c7626b901be78755666cad331f22566815195100f63f76

File: ./packages/contracts/contracts/L1/rollup/IChainStorageContainer.sol
SHA3: 84b56bd49d509b2a8013afce6676c788f6b2d3ed8b9dec0e5ad8722cf95cfd8a

File: ./packages/contracts/contracts/libraries/utils/Lib_Uint.sol
SHA3: e2f9edf1de8325ae0701cce2a358c05adccd24089f4c03b5db64f3d65aa16244
```

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 scoring methodology.

## Documentation quality

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

- Functional requirements are provided.
- Technical description is partially missed.

## Code quality

The total Code Quality score is **9** out of **10**.

- Most of the code follows best practices.
- There is redundant and commented code in the contracts.
- The development environment is configured.

## Test coverage

Test coverage of the project is **35%**.

- The libraries are covered with tests.
- The test coverage for main contracts is low, and some of the main tests do not run.

## Security score

As a result of the audit, the code contains **2** medium and **5** low severity issues. The security score is **8** 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: **5.9**.

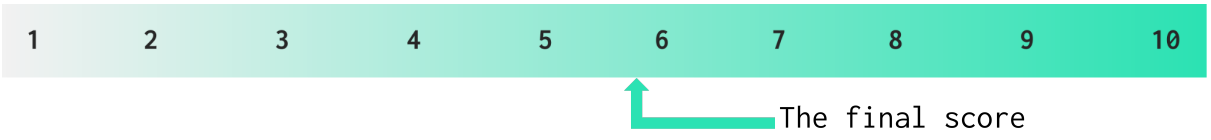


***Table. The distribution of issues during the audit***

| Review date | Low | Medium | High | Critical |
|---|---|---|---|---|
| 17 October 2022 | 18 | 5 | 9 | 2 |
| 01 November 2022 | 19 | 3 | 6 | 0 |
| 05 December 2022 | 5 | 2 | 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. | 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 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. | 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 identifiers should always be used. All parameters from the signature should be used in signer recovery | Not Relevant |
| **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. | Not Relevant |
| **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. | 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

*Metis* is an Ethereum layer 2 protocol, the audited part of the protocol is layer 1 (rollup and messaging with L2) with the following contracts:

- *ChainStorageContainer* – is a contract for storing transaction data:
  - transaction batches for the *Canonical Transaction Chain*,
  - chain state batches for the *State Commitment Chain*

  using *Lib_Buffer* arrays. There are data arrays in the contract for different chains and the defined default chain (1088). The contract functionality allows to push, set, get and delete transaction data from the defined index; set and update the global metadata (extra data in the *Lib_Buffer*) for each chain information array.
- *CanonicalTransactionChain* – is a contract that stores transactions that must be applied by the rollup state. The contract functionality allows MVM Sequencer to append the batches of the transactions (calldata is used to send and store the compressed transactions data) and to enqueue the transactions (which MVM Sequencer must eventually append to the rollup state.) through the L1CrossDomainMessenger contract.

  Transactions information is stored split for the different chains and the defined default chain (1088), there is a Sequencer for each chain.
- *StateCommitmentChain* – is a contract that stores the proposed state roots, as the results of each transaction in the Canonical Transaction Chain in the 1:1 correspondence.
  The contract functionality allows MVM Fraud Verifiers (for each chain) to delete state batches within the defined fraud proof window.

  The contract functionality allows to verify validity of the state within the provided inclusion proof.

  Transactions information is stored split for the different chains and the defined default chain (1088).
- *L1CrossDomainMessenger* – is a contract that allows communication between L1 and L2: whitelisted addresses to send messages from L1 to L2. The contract functionality allows relaying messages from L2 to L1 (The message is verified, checked for absence in the previously received and blocked by the owner messages) and to replay the message if it was rejected because of exceeding the L2 Gas limit.

  The contract logic allows the interaction with the different chains, the defined default chain is 1088.

  The contract has pausing functionality.
- *L1StandardBridge* – is a contract that provides the ability to deposit ETH and compliant ERC-20 tokens to L2 and withdraw them from L2. The bridged tokens and coins are stored in the contract. The contract functionality allows users to deposit tokens and coins for their own or other defined address. In case of transferring L1 Metis token, the

MVM_COINBASE coin will be bridged to L2 and vice versa for the withdrawal from L2.

For bridging tokens and coins, users pay the fee in ETH in the amount of Gas that will be used for L2 multiplied by the discount value defined in the MVM_DiscountOracle contract. Users define the Gas that will be used for L2, it is increased to the minimal Gas value defined in the MVM_DiscountOracle contract (200 000 initially) in case the user defines the less value.

- *MVM_DiscountOracle* – is a contract that manages the discount values for tokens and coins bridging, minimal Gas for L2 value, and the whitelist for sending messages to L2 (the contract provides the functionality to allow all addresses to send messages).
- *Lib_AddressManager* – is a contract that stores addresses by their names (converting names to hashes).
- *Lib_Buffer* – is a library that implements a bytes32 storage array: stores array items, length and extra data. The array functionality allows to push, set and obtain elements, delete elements after the defined index, set and get array extra data. The library is used in the *ChainStorageContainer* contract.
- *Lib_MerkleTrie* – is a library that provides the base Merkle Trie functions, used in the *Lib_SecureMerkleTrie* library.
- *Lib_SecureMerkleTrie* – is a library with the Merkle Trie functions, used in the *L1CrossDomainMessenger* contract for the relayed messages verifications.
- *Lib_RLPReader* – is a library with the functions for the RLP reading, used in the *Lib_MerkleTrie, Lib_CrossDomainUtils and Lib_OVMCodec* contracts.
- *Lib_RLPReader* – is a library with the functions for the RLP writing, used in the *Lib_MerkleTrie*, *Lib_CrossDomainUtils* and *Lib_OVMCodec* contracts.
- *Lib_MerkleTree* – is a library with the Merkle Tree implementation, used in the StateCommitmentChain for the state verification and the batch root obtaining.
- *Lib_BytesUtils* – is a library with the utility functions for the bytes, used in the *Lib_MerkleTrie* contract.
- *Lib_OVMCodec* – is a library with the utility hashing functions and transactions structs used in the *IL1CrossDomainMessenger*, *L1CrossDomainMessenger*, *CanonicalTransactionChain*, *ICanonicalTransactionChain*, *IStateCommitmentChain* and *StateCommitmentChain* contracts.
- *CrossDomainEnabled* – is a contract with the helper functions for cross-domain communications used in the *L1StandardBridge* contract.

- *Lib_CrossDomainUtils* – is a library with functions that generate the cross domain calldata for the messages, used in the *L1CrossDomainMessenger* contract.
- *Lib_PredeployAddresses* – is a library that stores some constant addresses, used in the *L1CrossDomainMessenger* and *L1StandardBridge* contracts.
- *Lib_DefaultValues* – is a library that stores *DEFAULT_XDOMAIN_SENDER* value, used in the *L1CrossDomainMessenger* contract.
- *Lib_AddressResolver* – is an abstract contract that allows to resolve addressed through the *Lib_AddressManager* contract, used in the *L1CrossDomainMessenger*, *CanonicalTransactionChain*, *ChainStorageContainer*, *StateCommitmentChain*, *BondManager* and *MVM_DiscountOracle* contracts.
- *Lib_Bytes32Utils* – is a library with the utility functions for the bytes32, imported in the *Lib_OVMCodec* contract.
- *AddressAliasHelper* – is a library with functions that allow to compute the L1 address from an L2 alias and vice versa, used in the *L1CrossDomainMessenger* and *CanonicalTransactionChain* contracts.
- *BondManager* – is a contract that verifies the proposers, used in the *StateCommitmentChain* contract.
- *IL1CrossDomainMessenger* – is an interface for the *L1CrossDomainMessenger*, inherited by the *L1CrossDomainMessenger* contract.
- *ICanonicalTransactionChain* – is an interface for the *CanonicalTransactionChain*, inherited by the *CanonicalTransactionChain* contract, used in the *StateCommitmentChain* contract.
- *IStateCommitmentChain* – is an interface for the *StateCommitmentChain*, inherited by the *StateCommitmentChain* contract, used in the *L1CrossDomainMessenger* contract.
- *iMVM_DiscountOracle* – is an interface for the *MVM_DiscountOracle*, inherited by the *MVM_DiscountOracle* contract, used in the *L1CrossDomainMessenger* and *L1StandardBridge* contracts.
- *IChainStorageContainer* – is an interface for the *ChainStorageContainer*, inherited by the *ChainStorageContainer* contract, used in the *CanonicalTransactionChain, IStateCommitmentChain, StateCommitmentChain* and *ICanonicalTransactionChain* contracts.
- *IBondManager* – is an interface for the *BondManager*, inherited by the *BondManager* contract, used in the *StateCommitmentChain* contract.
- *IL1StandardBridge* – is an interface for the *L1StandardBridge*, inherited by the *L1StandardBridge* contract.

- *IL1ERC20Bridge* – is an interface for the *L1StandardBridge* ERC-20 functions, inherited by the *IL1StandardBridge* contract, used in the *L1StandardBridge* contract.
- *ICrossDomainMessenger* – is an interface for the cross domain messengers, inherited by the *IL1CrossDomainMessenger* contract, used in the *CrossDomainEnabled* contract.
- *IL2ERC20Bridge* – is an interface for the L2 ERC-20 bridge functions, used in the *L1StandardBridge* contract.
- *Lib_Uint* – is a library that contains the function for converting uint to string, used in the project contracts.

## Privileged roles

- The owner of the *ChainStorageContainer* contract can set global metadata for each chain information array, set and push values to the arrays, and delete values from the array after the defined index.
- The *CanonicalTransactionChain* contract has the privileged roles of burn admin, Proxy__OVM_L1CrossDomainMessenger, OVM_Sequencer, MVM_Sequencer :
    - The burn admin can set the enqueue Gas parameters.
    - The Proxy__OVM_L1CrossDomainMessenger can enqueue transactions.
    - The OVM_Sequencer and MVM_Sequencer (of each chain) can append transaction batches.
    - The manager can push and set transactions to queue; push, set and delete transactions data and batch global metadata. (For each chain).
- The *StateCommitmentChain* contract has the privileged roles of METIS_MANAGER, MVM_FraudVerifier and proposers:
    - The METIS_MANAGER can set fraud proof window value.
    - The MVM_FraudVerifier (for each chain) can delete state batches.
    - The proposers can append state batches.
- The owner of the *L1CrossDomainMessenger* contract can pause the contract, block and allow the messages to be relayed from L2.
- The *CrossDomainAccount* of the *L1StandardBridge* contract can finalize the tokens and coins withdrawal from L2 to L1.
- The *manager* of the *MVM_DiscountOracle* contract can set the discount, minimal L2 Gas values, manage them for sending messages to L2 whitelist, or allow all users to send messages, withdraw native coins to the
- The owner of the *Lib_AddressManager* contract can add and modify the addresses stored.

## Risks

- There are contracts in the repository, and the off-chain project logic, not included in the audit scope, their secureness and the secureness of all the protocol cannot be verified.

# Findings

## ■■■■ Critical

### 1. Funds Lock

The fees are transferred to the *MVM_DiscountOracle* contract through the *processL2SeqGas* function, but there is no mechanism for their withdrawal.

Therefore, the fees will be locked in the contract.

**Path:** ./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol : processL2SeqGas()

**Recommendation**: add the possibility of coins withdrawal from the *MVM_DiscountOracle* contract.

**Status**: Fixed (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

### 2. Denial of Service Vulnerability

Both *relayMessage* and *relayMessageViaChainId* use a *nonReentrant* modifier.

Because *relayMessageViaChainId* is called in the *relayMessage*, the *relayMessage* is inoperable.

**Path:** ./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : relayMessage(), relayMessageViaChainId()

**Recommendation**: remove the *nonReentrant* modifier from the *relayMessage* function.

**Status**: Fixed (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

## ■■■ High

### 1. Denial of Service Vulnerability

The *_chainId* and *_totalElements* parameters are used vice versa when emitting the *SequencerBatchAppended* events in the *appendSequencerBatch* function.

According to the documentation, the *SequencerBatchAppended* event is detected by the client and processed. Due to this, incorrect data in the event will lead to the incorrect data analysis, may block the system, or make it work in an incorrect way.

**Path:** ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : appendSequencerBatch()

**Recommendation**: use the correct data when emitting the event.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

2. **Requirements Violation**

According to the documentation, any account may send message from L1 to L2, but there is a whitelist check for the *L1CrossDomainMessenger.sendMessage* and *L1CrossDomainMessenger.sendMessageViaChainId* functions (*MVM_DiscountOracle.processL2SeqGas*).

Therefore, the contract functionality provides the ability to restrict access to the messages sending and the requirements can be violated.

**Path:**
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : sendMessage(), sendMessageViaChainId()

**Recommendation**: implement the code according to the requirements or change the requirements to match the code.

**Status**: Fixed (updated documentation) (Revised commit: 3e388ada0011bf6aefbf0085a8df02ad674cc548)

3. **Access Control Violation; Replay Attack Vulnerability**

The *proposer* value that is used for obtaining the proposer address is defined by the user through the *StateCommitmentChain.appendStateBatchByChainId* function parameter.

Therefore, any address that was added to the *Lib_AddressResolver* will be able to append the state batches.

Collateralizing validation in the *BondManager.isCollateralizedByChainId* function is performed without the chain splitting.

Due to this, addresses that are collateralized for one chain may have an access to appending the state batches for all the chains.

**Paths:**
./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : appendStateBatchByChainId(), _appendBatchByChainId()

./packages/contracts/contracts/L1/verification/BondManager.sol : isCollateralizedByChainId()

**Recommendation**: ensure that only correct proposers can have access to the state appending, clarify the collateralizing validation requirements and implement the code according to them.

**Status**: Fixed (the chain id check is performed in *StateCommitmentChain.appendStateBatchByChainId* function, *BondManager.isCollateralizedByChainId* function does not have the

chain splitting) (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

4. **Denial of Service Vulnerability**

   The contract has a pausing functionality, but it is impossible to unpause it.

   Therefore, the contract may become inoperable.

   **Path:**
   ./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : pause()

   **Recommendation**: add the ability to unpause the contract.

   **Status**: Fixed (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

5. **Requirements Violation; Data Consistency**

   According to the contract description, the CanonicalTransactionChain contract is "*an append-only log of transactions*" and the batches may be added by the Sequencer. The user with the manager role may update, delete and add the transactions data, update queue data.

   Therefore, the requirements are violated. Deleting and updating the transactions data may lead to storing inconsistent transaction information. Updating the queue data may block or change the transactions that users enqueued.

   **Path:**
   ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : setBatchByChainId(), deleteBatchElementsAfterInclusiveByChainId(), pushBatchByChainId(), setBatchGlobalMetadataByChainId(), setQueueByChainId()

   **Recommendation**: implement the code according to the requirements, ensure that transactions data is consistent.

   **Status**: Fixed (added comment with the behavior description and updated documentation) (Revised commit: 3e388ada0011bf6aefbf0085a8df02ad674cc548)

6. **Data Consistency; Requirements Violation**

   According to the documentation and contract description in the code, the *ChainStorageContainer* stores queued transactions. The *CanonicalTransactionChain.queue()* functions return the *ChainStorageContainer* contract. However, the queued transactions are added to the *CanonicalTransactionChain.queueElements* mapping.

   Therefore, the *CanonicalTransactionChain* contract does not store the queue, and the requirements are violated.

   **Paths:**
   ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : queue(), queueElements

./packages/contracts/contracts/L1/rollup/ChainStorageContainer.sol

**Recommendation**: ensure that queue storage is consistent and implemented according to the requirements.

**Status**: Fixed (removed redundant code and updated documentation) (Revised commit: 3e388ada0011bf6aefbf0085a8df02ad674cc548)

### 7. Requirements Violation

According to the contract description, any account should be able to enqueue the transaction. (Comment: "*The CTC also allows any account to 'enqueue' an L2 transaction*"). However, the *enqueueByChainId* function requires that the *msg.sender* is equal to the *Proxy__OVM_L1CrossDomainMessenger* address.

Therefore, it is only possible to enqueue transactions through the cross-domain messenger, and the requirements are violated.

The contract contains the Gas cost for the calling enqueue function and the ability set up in the constructor and can be changed through the *setGasParams* function. However, the enqueue functions (*enqueue*, *enqueueByChainId*) do not perform these Gas checks.

Due to this, the contract is vulnerable to spam attacks.

**Path:**
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : enqueueByChainId(), setGasParams(), constructor() enqueueGasCost, l2GasDiscountDivisor, enqueueL2GasPrepaid

**Recommendation**: clarify the requirements and implement the code according to them.

**Status**: Fixed (updated documentation, added spam checks) (Revised commit: 3e388ada0011bf6aefbf0085a8df02ad674cc548)

### 8. Requirement Violation

The comment in the *appendSequencerBatchByChainId* describes the Merkle root functionality that is not implemented. (*For efficiency reasons getMerkleRoot modifies the `leaves` argument in place while calculating the root hash therefore any arguments passed to it must not be used again afterwards*).

The comment in the *enqueueByChainId* function describes the queue data structure functionality that is not implemented. (*The underlying queue data structure stores 2 elements per insertion, so to get the real queue length we need to divide by 2 and subtract 1.*). The queue data structure stores 1 element per insertion and the queue data length is not divided by 2.

This may indicate that the code is not finalized and the requirements are violated.

**Path:**
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : appendSequencerBatchByChainId(), enqueueByChainId()

**Recommendation**: clarify the requirements and implement the code according to them.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

9. **Undocumented Functionality**

Bridging tokens and coins require paying fees, and there is the defined minimal L2 Gas limit, which may make the defined by user Gas value increase.

Such functionality should be described in the documentation.

**Path:**
./packages/contracts/contracts/L1/messaging/L1StandardBridge.sol : _initiateETHDepositByChainId(), _initiateERC20DepositByChainId()

**Recommendation**: describe the fees and minimal Gas value in the documentation.

**Status**: Fixed (updated documentation) (Revised commit: 3e388ada0011bf6aefbf0085a8df02ad674cc548)

## Medium

1. **Unfinalized Code**

The *registerSequencerByChainId* function does not perform any logic.

The code is not finalized.

**Path:** ./packages/contracts/contracts/L1/verification/BondManager.sol : registerSequencerByChainId()

**Recommendation**: finalize the code.

**Status**: Fixed (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

2. **Denial of Service Vulnerability; Unfinalized Code**

The *appendStateBatch* function is inoperable.

This may indicate that the code is not finalized.

**Path:**
./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : appendStateBatch()

**Recommendation**: finalize the code, remove the redundant code.

**Status**: Fixed (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

3. **Requirement Violation; Redundant Code**

   The functions are split by the chains by titles and contain the *_chainId* parameters, but the functionality they perform is not split.

   This may indicate that the requirements are violated, or the contracts contain redundant code.

   **Paths:**

   ./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : _verifyStorageProofByChainId()

   ./packages/contracts/contracts/L1/verification/BondManager.sol : isCollateralizedByChainId()

   ./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : insideFraudProofWindowByChainId(), _makeBatchExtraDataByChainId()

   ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : _makeBatchExtraDataByChainId(), _getBatchContextByChainId()

   **Recommendation**: clarify the requirements and implement the code according to them, remove the redundant code.

   **Status**: Reported

4. **Missing Index Validation**

   Function *set* does not check if there was a value set on the defined index (*_index*) and does not increment the buffer length.

   Therefore, the function will not revert on the setting value to the unexisting index.

   **Path:** ./packages/contracts/contracts/libraries/utils/Lib_Buffer.sol : set()

   **Recommendation**: check if the index exists when setting a value to it.

   **Status**: Fixed (Revised commit: defb4e1dbfa85cf9f8a248ea838b8dacf19860fa)

5. **Tests Failing**

   107 tests are failing, and the OVM tests do not run.

   **Recommendation**: ensure that all the tests run and pass.

   **Status**: Reported. 79 tests failed.

6. **Funds Lock**

   The fees are withdrawn to the different *MVM_Sequencer_Wrapper* for each chain, but the amounts of received fees for each chain are not stored in the contracts.

   Therefore, incorrect amounts can be transferred to the *MVM_Sequencer_Wrappers*.

**Path:** ./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol : withdrawToSeq()

**Recommendation**: store the received fees for each chain and do not allow *MVM_Sequencer_Wrapper* to withdraw more than was received for the corresponding chain.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

## Low

### 1. Redundant Functions

There are functions in the project that are never used.

**Paths:**

./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol : uint2str()

./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : _getBatchExtraData(), _makeBatchExtraData()

./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : _verifyXDomainMessage(), _verifyStateRootProof()

./contracts/libraries/trie/Lib_SecureMerkleTrie.sol : update(), getSingleNodeRootHash()

./contracts/libraries/trie/Lib_MerkleTrie.sol : update(), getSingleNodeRootHash()

**Recommendation**: remove the redundant code.

**Status**: Reported

### 2. Boolean Equality

Boolean constants can be used directly and do not need to be compared to *true* or *false*.

**Paths:**
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : relayMessageViaChainId(), _verifyStorageProof(), _verifyStorageProofByChainId(), _verifyStateRootProofByChainId()

./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol : isXDomainSenderAllowed()

**Recommendation**: use the boolean values directly.

**Status**: Reported. *L1CrossDomainMessenger.relayMessageViaChainId* is not fixed.

### 3. Unused Variables

There are never used variables in the contract.

**Path:**
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : BATCH_CONTEXT_LENGTH_POS, TX_DATA_HEADER_SIZE, BYTES_TILL_TX_DATA

**Recommendation**: remove the redundant variables.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

4. **Floating and Outdated Pragma**

The project contracts use floating pragma, and some of them use outdated pragma.

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.

**Paths:** All contracts

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

**Status**: Reported

5. **Code Duplication**

There are functions in the project that execute the same logic or contain the same code.

**Paths:**
./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : _makeBatchExtraDataByChainId(), _makeBatchExtraData()

./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : _getBatchExtraDataByChainId(), _getBatchExtraData()

**Recommendation**: reuse duplicated code.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

6. **Duplicate Code**

Duplication of code may lead to unnecessary Gas consumption and cause maintenance issues due to modifications in multiple places.

*sendMessage* and *sendMessageViaChainId* mostly identical. It is recommended to introduce the internal *_sendMessage* function with *chainId* parameter. *getQueueLengthByChainId* and *_sendXDomainMessageViaChainId* functions are able to work with *DEFAULT_CHAINID*

**Path:** ./contracts/L1/messaging/L1CrossDomainMessenger.sol: sendMessage(), sendMessageViaChainId()

**Recommendation**: reuse duplicated code.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 7. Incorrect Title

The role is obtained using the *OVM_Sequencer* title, but the project Sequencer is titled *MVM_Sequencer*.

**Path:** ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : appendSequencerBatch()

**Recommendation**: replace the role title with the correct one.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 8. Redundant Functionality

Obtaining the next element to be enqueued id is performed by calling the *_getBatchExtraDataByChainId* function, but it can be directly taken from the *_nextQueueIndex* mapping.

**Path:** ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : getNextQueueIndexByChainId()

**Recommendation**: remove the redundant functionality.

**Status**: Reported

### 9. Redundant Parameter

The *_chainId* parameter is not used in the functions.

**Paths:** ./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : _getBatchContextByChainId(), _makeBatchExtraDataByChainId()

./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : _makeBatchExtraDataByChainId()

**Recommendation**: remove the redundant functionality.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 10. Duplicate Code

Duplication of code may lead to unnecessary Gas consumption and cause maintenance issues due to modifications in multiple places.

*uint2str* function is duplicated across multiple contracts.

**Paths:** ./contracts/L1/messaging/L1StandardBridge.sol: uint2str()

./contracts/L1/rollup/StateCommitmentChain.sol: uint2str()

./contracts/L1/rollup/CanonicalTransactionChain.sol: uint2str()

./contracts/MVM/MVM_DiscountOracle.sol : uint2str()

**Recommendation**: move the function to library contract and reuse.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 11. Redundant Code

Both *deleteStateBatchByChainId* and *_deleteBatchByChainId* perform the batch header validity check.

The redundant code increases the Gas consumption and decreases the code readability.

**Path:**
./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : deleteStateBatchByChainId(), _deleteBatchByChainId()

**Recommendation**: remove the redundant code.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 12. Redundant Code

The *_appendBatchByChainId* function contains the functionality for the case when the *msg.sender != proposer*, but is redundant as the *msg.sender* is always equal to the *proposer* due to the collateralizing check in the *appendStateBatchByChainId* function.

The redundant code increases the Gas consumption and decreases the code readability.

**Paths:**
./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : _appendBatchByChainId()

**Recommendation**: remove the redundant code.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 13. Redundant “payable” Keyword

The functions are marked with the *payable* keyword, but they do not execute logic for the native coin.

The redundant code increases the Gas consumption and decreases the code readability.

**Paths:**
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : replayMessage(), replayMessageViaChainId()

**Recommendation**: remove the redundant code.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 14. Default Visibility Usage

Variables` visibilities are not specified. Specifying state variables` visibility helps to catch incorrect assumptions about who can access the variable.

**Paths**: ./packages/contracts/contracts/MVM/MVM_DiscountOracle.sol : allowAllXDomainSenders

./contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : queueElements

./contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : DEFAULT_CHAINID

**Recommendation**: define the visibilities explicitly.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 15. Redundant Operation

*_initiateETHDepositByChainId* function allows depositing nothing in cases when *fee == msg.value*, which make no sense.

**Path**: ./contracts/L1/messaging/L1StandardBridge.sol: _initiateETHDepositByChainId()

**Recommendation**: use *require(fee < msg.value)* instead (not *<=*).

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 16. Incorrect Event Emit

*_initiateETHDepositByChainId* function emits *ETHDepositInitiated* with *msg.value*, but *msg.value - fee* should be emitted instead.

**Path**: ./contracts/L1/messaging/L1StandardBridge.sol: _initiateETHDepositByChainId()

**Recommendation**: emit the correct deposit value.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 17. Unused Imports

There are imports in the contract that are not used.

**Paths**:
./packages/contracts/contracts/libraries/codec/Lib_OVMCodec.sol : import { Lib_RLPWriter } from "../rlp/Lib_RLPWriter.sol";

import { Lib_BytesUtils } from "../utils/Lib_BytesUtils.sol";

```
import { Lib_Bytes32Utils } from "../utils/Lib_Bytes32Utils.sol";
```

**Recommendation**: remove the redundant imports

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 18. 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:**
./packages/contracts/contracts/L1/messaging/L1CrossDomainMessenger.sol : xDomainMessageSender(), sendMessage(), sendMessageViaChainId(), relayMessage(), replayMessage()

./packages/contracts/contracts/L1/rollup/CanonicalTransactionChain.sol : queue(), getTotalBatches(), getNextQueueIndex(), getLastTimestamp(), getLastBlockNumber(), getQueueElement(), getNumPendingQueueElements(), getQueueLength(), getTotalBatchesByChainId(), getNextQueueIndexByChainId(), getLastTimestampByChainId(), getLastBlockNumberByChainId(), getQueueElementByChainId(), getNumPendingQueueElementsByChainId(), getQueueLengthByChainId(), appendSequencerBatchByChainId(), pushQueueByChainId(), setQueueByChainId(), setBatchGlobalMetadataByChainId(), getBatchGlobalMetadataByChainId(), lengthBatchByChainId(), pushBatchByChainId(), setBatchByChainId(), getBatchByChainId(), deleteBatchElementsAfterInclusiveByChainId()

./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : setFraudProofWindow(), getTotalElements(), getTotalBatches(), getLastSequencerTimestamp(), appendStateBatch(), deleteStateBatch(), verifyStateCommitment(), insideFraudProofWindow(), appendStateBatchByChainId()

**Recommendation**: use the external visibility for functions never called from the contract.

**Status**: Fixed (Revised commit: 756a26d8651a5da3c438391d7854bbf4a2ff181a)

### 19. Redundant Code

The *appendStateBatchByChainId* function contains the *proposer* parameter, which is validated for the equality with *msg.sender* and *string(abi.encodePacked(uint2str(_chainId),"_MVM_Proposer")))*.

Therefore, this functionality can be simplified to equality checking of *msg.sender* and *string(abi.encodePacked(uint2str(_chainId),"_MVM_Proposer")))*.

**Path:**
./packages/contracts/contracts/L1/rollup/StateCommitmentChain.sol : appendStateBatchByChainId()

**Recommendation**: remove the redundant code.

**Status**: Reported

# 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.