SIDECHAIN BRIDGE SECURITY ASSESSMENT

March 23, 2022

Prepared For:

Rob Kornacki, Milkomeda Foundation

Prepared By:

John Bird

Chris Masden

TABLE OF CONTENTS

VULNERABILITY STATISTICS

SEVERITY BREAKDOWN

Severity	**Count**
Critical	0
High	0
Medium	3
Low	5
Note	9
Total	17

FIXES SUMMARY

Finding	**Severity**	**Status**
M1	Medium	Fixed in pull request #816
M2	Medium	Not an issue
M3	Medium	Fixed in pull request #817
L1	Low	Fixed in pull request #887
L2	Low	Not fixed
L3	Low	Not fixed
L4	Low	Fixed in pull requests #818, #821, and #886
L5	Low	Fixed in pull request #874
N1	Note	Fixed in pull request #883
N2	Note	Fixed in pull request #819
N3	Note	Not fixed
N4	Note	Not fixed. Acknowledged in pull request #854
N5	Note	Fixed in pull request #856
N6	Note	Fixed in pull request #891
N7	Note	Fixed in pull request #853
N8	Note	Not fixed
N9	Note	Fixed in pull request #820

EXECUTIVE SUMMARY

This report contains the results of Arbitrary Execution's assessment of the Milkomeda sidechain bridge smart contracts. The Milkomeda protocol provides cross-chain interoperability between two blockchains. The sidechain bridge handles logic on the EVM side of the bridge.

Two Arbitrary Execution engineers conducted this review over a 2-week period, from February 7, 2022 to February 18, 2022. The audited tag was `v1.1.0-rc1` (commit hash `860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a`) in the `dcSpark/milkomeda-validator` repository. The Solidity files in scope for this audit included all contracts in the `m1/contracts` directory with the exception of contracts in the `dev` directory, and `Migrations.sol` - which is an unused contract in production. The complete list of files is in Appendix B.

Our efforts were focused on the proxy-implementation model for the bridge contract, as well as how the bridge handles users' funds. These contracts will be deployed on custom Hyperledger Besu networks, so differences from Ethereum mainnet had to be accounted for.

The assessment resulted in 17 findings ranging in severity from medium to note (informational). One of the medium findings identified a situation where `wADA` can become trapped in the implementation contract. The other medium findings involve contract deployment and removing validator nodes from the network. The low findings include an issue with the proxy-implementation deployment and unsafe external calls. The note findings contain some observations we felt necessary to highlight, typographical suggestions, and opportunities for gas optimizations.

The Milkomeda bridge contracts are separated into logical components that contain NatSpec and other in-line comments. The code is well documented and straightforward to follow. High-level documentation is present and clear as well. The smart contract repo also contains a Mocha unit test suite that exercises bridge functionality.

Update

At the time of the fix review, 11 findings were resolved according to our recommendations and 1 finding was identified as a non-issue. The 5 remaining findings were not fixed. The Milkomeda team acknowledges these findings and intends to investigate them at a later date.

FINDINGS

MEDIUM SEVERITY

[M1] IMPLEMENTATION CONTRACT SHOULD NOT BE ABLE TO RECEIVE WADA

The implementation contract `Multisig.sol` has a receive function that will accept `wADA` but the `wADA` that is sent to the contract cannot be withdrawn. All interactions should be sent via `Proxy.sol` which can properly receive all of the `wADA`.

RECOMMENDATION

Remove the `receive` function in `Multisig.sol`

UPDATE

Fixed in pull request #816.

[M2] NO CHECKS ON MINIMUM VALIDATOR COUNT

Milkomeda runs a network for its sidechain that uses an Istanbul Byzantine Fault Tolerant (IBFT) consensus mechanism. Specifically, the Hyperledger Besu network that the Milkomeda Bridge will be deployed on uses IBFT 2.0.

IBFT 2.0 requires a minimum of 4 validators to remain Byzantine fault tolerant. removeValidator has no checks on validator count, and the `validRequirement` modifier only checks if `validatorCount` is nonzero. Validators can currently vote to remove a validator and fall below this threshold. If the network is no longer Byzantine fault tolerant, the network may not function correctly and reach consensus despite nodes failing or propagating incorrect information to peers.

RECOMMENDATION

Consider updating the `validRequirement` modifier to check if `validatorCount < 4`.

UPDATE

Not an issue. Milkomeda’s statement:

We do not think this is something we can check. It makes sense you spotted this but in essence there is no minimum validator to have in the smart contract in relation to the IBFT requirement. The smart contract has a set of validators that is maintained in a different list than the IBFT. It happens to be the same person at the moment but there's no hard requirement to make the IBFT validator the same as the Milkomeda Validator.

[M3] LACK OF ADDRESS SANITY CHECKS

In `Proxy.sol` the constructor assigns the implementation storage variable without checking that it is non-zero and without checking that a smart contract exists at that address.

In `Multisig.sol` the upgradeContract function changes the `implementation` storage variable without performing the above mentioned checks.

In the event of a bad address being used when deploying the proxy, a simple redeployment of the proxy would fix this issue. In the case of upgrading from one implementation to another, a more serious consequence could occur if a bad address is used. The funds (`wADA, ERC20, ERC721, ERC1155`) would become frozen inside of the proxy contract.

RECOMMENDATION

Due to the importance of the `implementation` variable, consider adding the OpenZeppelin `isContract` modifier on the address that `implementation` will be set to. The `isContract` modifier would ensure that the address is non-zero and there is deployed bytecode at the address.

UPDATE

Fixed in pull request #817.

LOW SEVERITY

[L1] IMPLEMENTATION SET BUT NOT INITIALIZED

Upon deployment, the proxy sets the implementation address, but does not initialize it. The initialization of the contract is vulnerable to front running and a malicious attacker could initialize the contract with the settings they choose. The proxy would then have to be redeployed as there is no way to change the `implementation` variable inside of the proxy once set by the constructor.

RECOMMENDATION

Consider calling the initialize function in the constructor of `Proxy.sol` so that the implementation address cannot be set without also initializing the implementation contract.

UPDATE

Fixed in pull request #887

[L2] LACK OF EVENT EMISSIONS

`Multisig.sol` contains various administrative functions that represent important state changes in the contract. Certain functions do not emit events when sensitive changes are made.

RECOMMENDATION

Consider adding the following events to aid with tracking and for notifying off-chain observers:

```
AddedValidator
RemovedValidator
NewQuorum
NewStargateAddress
ContractUpgraded
TransactionRemoved
TransactionAdded
```

UPDATE

Not fixed. Milkomeda's statement for this issue:

> *Acknowledged. We will investigate this at a later time.*

[L3] POSSIBLY DANGEROUS DISTRIBUTION OF REWARDS

In `Rewards.sol` the `withdrawRewards` function has a possible dangerous distribution method for sending rewards to each validator by using the following code. The danger is that a validator could be a smart contract and this is acknowledged in the comments. However, the risk is still present.

RECOMMENDATION

Consider using the OpenZeppelin `isContract` modifier on the `addValidator` function inside of `Multisig.sol`. This would drastically reduce the chance of a smart contract being added as a validator.

UPDATE

Not fixed. Milkomeda's statement for this issue:

> *Acknowledged. We will investigate this at a later time.*

[L4] UNIMPLEMENTED TODOS

There are TODOs in `Multisig.sol`, `SidechainBridge.sol`, and `Types.sol`. Three in particular are worth highlighting:

- removeValidator does not clear validator votes on pending proposals
- There is no functionality to migrate proposals in the event a validator provides bad data in a proposal
- Tokens are not returned to users if an unwrapping request gets voted out

If these TODOs are overlooked, there is a risk that deployed code does not match the design specification of the protocol. Implementing these features will reduce the impact of a malicious validator, and allow users to recover funds from the bridge.

RECOMMENDATION

Consider implementing TODOs or documenting reasons for not doing so.

UPDATE

- `removeValidator` does not clear votes

  TODO removed in pull request #818. Milkomeda's statement for this issue:

We do not need to remove validator votes from unfinished proposals. At the time of the vote the validator submitted its vote as part of being in the protocol validator. If we have removed this validator, the proposal will still require at least another validator to vote for this proposal.

1. *The proposal has received `quorum - 1` vote before one of the validators is removed from the list. assuming the quorum remain the same, the proposal still requires `1` more vote anyway and has been validated by all the other participants.*
2. *The proposal has received `quorum - 1` vote before the `quorum` is reduced by `1`. The proposal will still require an extra vote in order to be detected that the `quorum` has been reached.*
3. *The quorum is already reached: the proposal is already processed*
4. *the proposal has received `quorum - k` votes. The proposal will still need to receive at least `k` extra votes to be processed.*

In short, we rely on the honest majority to continuously guarantee that the proposals being voted on are necessary. I.e. if a bad actor who was voting on invalid proposal is being removed, we assume the honest majority will not vote for invalid proposals anyway.

- Proposal migration

  Fixed in pull request #821.

- Tokens are not returned to users if an unwrapping request gets voted out

  TODO removed in pull request #886. Milkomeda's statement for this issue:

  We do not believe the transaction id can be verified on the smart contract side. It is not possible for the sidechain to actually verify a transaction on the mainchain (Cardano here) has happened successfully. However, the validator will know this. If the transaction that was proposed in the smart contract becomes invalid and is not included in the blockchain then the validator will initiate a migration.

[L5] `VOTEFORTRANSACTION` CAN FAIL WITHOUT ANY INDICATION

In `Multisig.sol` the `voteForTransaction` function checks to see if a call to `confirmTransaction` should be issued. There already exists a revert statement that will cause the transaction to revert if a `transactionId` already exists and the parameters submitted do not match the parameters stored for the given transaction.

However, if `voteForTransaction` is called with parameters that do match a transactionId but the transaction has already been executed, `voteForTransaction` will succeed which can lead to a user thinking the vote has been cast when it actually hasn't.

RECOMMENDATION

When checking if a `transactionId` has been executed, consider inserting a `revert` statement that will cause a transaction to be reverted in this situation.

UPDATE

Fixed. Event emissions were adjusted in pull request #874. Milkomeda's statement for this issue:

> *A transaction will be executed as soon as the quorum is reached. However, it is possible that other validators votes are yet to be registered by the smart contract. I.e. all the validators will check independently from each other if an action can be done and will all vote as soon as possible to execute a transaction. If we were to throw an error on votes that are not necessary (but not invalid) we would have up to `validators - quorum` errors reported to all the nodes. This will create unnecessary noise. Once the transaction has been executed all the validators are notified by an event that the transaction was executed.*

NOTE SEVERITY

[N1] FUNCTIONS CAN BE MARKED EXTERNAL

Due to the low level call nature of transaction execution, the following list of functions can be marked `external` which will result in gas savings.

```
Multisig.sol
-----------------
addValidator
removeValidator
replaceValidator
upgradeContract
voteForTransaction
removeTransaction
getConfirmationCount
getTransactionCount
getValidators
getConfirmations
getTransactionIds

Rewards.sol
-----------------
withdrawRewards

SidechainBridge.sol
-----------------
initialize
submitUnwrappingRequest
submitUnwrappingProposalTransaction
voteOnUnwrappingProposalTransaction
updateProtocolMagic
updateBridgeParameters
getUnwrappingProposalRequest
getUnwrappingProposalTransaction
getUnwrappingProposalTransactionWitness
onERC1155Received
onERC1155BatchReceived

TokenRegistry.sol
-----------------
addAssetToRegistry
updateAsset
removeAssetFromRegistry
getAssetIdsCount
getAssetIds
findAssetIdByAddress
```

RECOMMENDATION

Consider changing the visibility of the aforementioned functions from `public` to `external`.

UPDATE

Fixed in pull request [#883](#).

[N2] INCORRECT REQUIRE STATEMENT

In `Multisig.sol` the function `executeTransaction` has a [require statement](#) that can never fail. `value` is an unsigned integer which will always be greater than or equal to 0. If `transaction.value` is less than the `WRAPPING_FEE`, then the transaction will revert since the `solc` compiler is version 0.8.x.

RECOMMENDATION

Consider removing the require statement and examine if this behavior is intentional.

UPDATE

Fixed in pull request [#819](#). The original issue is no longer present. The team added a new require statement that checks `value` against `WRAPPING_FEE`.

[N3] CONTRACTS NOT OPTIMIZED BEFORE DEPLOYMENT

The implementation and proxy contracts are not optimized before deployment.

RECOMMENDATION

To reduce the size of deployed bytecode and to reduce the cost of executing transactions, consider running the `solc` optimizer.

UPDATE

Not fixed. Milkomeda's statement for this issue:

> *Acknowledged. We will investigate this at a later time.*

[N4] INCONSISTENT USE OF TOKEN RECEIVER CONTRACTS

The `SidechainBridge` contract inherits from ERC1155Receiver and overrides the onERC1155Received and onERC1155BatchReceived functions. This is good behavior and allows the contract to use the `safeTransferFrom` function on `ERC1155` tokens. The `ERC721` tokens currently use `transferFrom` because the `SidechainBridge` contract does not inherit from the `ERC721Receiver` contract.

RECOMMENDATION

Consider having `SidechainBridge` inherit from `ERC721Receiver` and override the `onERC721Received` function. Then in `SidechainBridge.sol` the `safeTransferFrom` function can be used to move `ERC721` tokens from `msg.sender` to the contract address.

UPDATE

Not fixed. Milkomeda added a comment with their reasoning in pull request #854:

```
// we can safely use transferFrom instead of safeTransferFrom
// because we are always transferring to ourselves
```

[N5] NOT FOLLOWING CHECKS-EFFECTS-INTERACTIONS PATTERN

`submitUnwrappingRequest` makes external calls to allowed assets before modifying state variables. A malicious token contract could hijack control flow with the `transferFrom` and `safeTransferFrom` function calls. The risk of reentrancy exists in the `submitUnwrappingRequest` function, however it would not have any security impact because the malicious asset must first be voted in by a validator majority. Furthermore, an attacker is limited in what functions can be called because of function modifiers on other public methods.

RECOMMENDATION

While there is no impact in this particular case, it is best practice to follow the checks-effects-interactions pattern.

Consider moving external calls in `submitUnwrappingRequest` below the state changes on lines 207-212

UPDATE

Fixed in pull request #856.

[N6] NOT FOLLOWING NAMING CONVENTIONS

Parameters for initialize that have a corresponding storage variable are prefixed with an underscore.

`_minWmainUnwrap` does not have a corresponding storage variable, so there is no need for the underscore.

RECOMMENDATION

Consider removing the underscore from `_minWmainUnwrap` for clarity.

UPDATE

Fixed in pull request #891.

[N7] UNSTRUCTURED STORAGE PROXIES

In `Proxy.sol` the first storage slot is used to store the implementation address. This can be error-prone because the proxy performs delegate calls on an `implementation` address and may conflict with a storage slot on the implementation contract. Storage collision can occur (but does not in the current code) if the proxy and implementation do not share the same storage patterns. EIP-1967 was created to help address this issue and it uses a standardized storage slot location for the `implementation` storage variable. This will also allow the implementation contract to declare and use storage variables regardless of the storage layout in the proxy, with one exception: The implementation contract is responsible for updating the `implementation` address and as such will have to use the standardized storage slot for the `implementation` variable as the proxy does.

RECOMMENDATION

Consider using the EIP-1967 standardized storage slot for the `implementation` storage variable. More info can be found here: unstructured-storage-proxies

UPDATE

Fixed in pull request #853.

[N8] GAS SAVINGS

Three areas were identified where gas efficiency can be improved.

CACHE `ARRAY.LENGTH` VALUES

There are several instances where `array.length` is computed inside `for` loops or multiple times in the same function.

RECOMMENDATION

Consider caching array lengths whenever possible, and updating the following:

`Multisig.sol`:

- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L101
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L103
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L124
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L232
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L234
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L245
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L300
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L313
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L335

- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L338
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L359
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L360
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L363
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Multisig.sol#L366

`Rewards.sol:`

- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Rewards.sol#L18
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Rewards.sol#L19
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/Rewards.sol#L21

`SidechainBridge.sol:`

- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/SidechainBridge.sol#L120

`TokenRegistry.sol:`

- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/TokenRegistry.sol#L48
- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/TokenRegistry.sol#L50

- https://github.com/dcSpark/milkomeda-validator/tree/860d5c4cb34e3a5ed6bfcd6d22e67010dd80e02a/m1/contracts/src/TokenRegistry.sol#L76

ONLY CALL `CHANGEQUORUM` WHEN NECESSARY

In `Multisig.sol`, the addValidator and removeValidator functions always call `changeQuorum` regardless of the function parameters.

If `newStargateAddress` and `newQuorum` match the current `stargateAddress` and `quorum`, this call is unnecessary.

RECOMMENDATION

Consider adding logic in `addValidator` and `removeValidator` to check if `quorum` and `stargateAddress` will remain unchanged.

SIMPLIFY `VALIDREQUIREMENT` MODIFIER

The if statement in `validRequirement` in its current state can be simplified because there are no cases where `validatorCount` can be 0 when `quorum` is non-zero.

RECOMMENDATION

Consider simplifying the `if` statement.

Note that if the `validatorCount` were to be checked against a number other than zero (e.g. `validator count < 4`, as mentioned in No checks on minimum validator count), this check cannot be removed.

UPDATE

Not fixed. Milkomeda's statement for this issue:

> *Acknowledged. We will investigate this at a later time.*

[N9] TYPOGRAPHICAL ERRORS

There are code comments that were not updated when variables were renamed.

- A comment in `Rewards.sol` was not updated when `WITHDRAWAL_EPOCH` was renamed to `WITHDRAWAL_PERIOD`

- Comments in Types.sol and State.sol were not updated when `WMAIN_RELEASE_FEE` was renamed to `WRAPPING_FEE`
- `Multisig.sol` has a minor typo with the word "filter"

RECOMMENDATION

Consider updating the comments to match the referenced variables and fixing typos:

- `WITHDRAWAL_EPOCH` -> `WITHDRAWAL_PERIOD`
- `WMAIN_RELEASE_FEE` -> `WRAPPING_FEE`
- `wmainReleaseFee` -> `WRAPPING_FEE`
- `filer` -> `filter`

UPDATE

Fixed in pull request #820.

APPENDIX A – SEVERITY DEFINITIONS

Critical	This issue is straightforward to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
High	This issue is difficult to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
Medium	This issue is important to fix and puts a subset of users' data at risk and is possible to lead to moderate financial impact.
Low	This issue is not exploitable on a recurring basis and cannot have a significant impact on execution.
Informational (Note)	This issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this audit.

APPENDIX B – LIST OF FILES IN SCOPE

```
Milkomeda-validator/m1/contracts/src/proxy/Proxy.sol

Milkomeda-validator/m1/contracts/src/Multisig.sol

Milkomeda-validator/m1/contracts/src/Rewards.sol

Milkomeda-validator/m1/contracts/src/SidechainBridge.sol

Milkomeda-validator/m1/contracts/src/State.sol

Milkomeda-validator/m1/contracts/src/TokenRegistry.sol

Milkomeda-validator/m1/contracts/src/Types.sol
```